Agent Organization in the Knowledge Plane

by

Ji Li

B.S., Nanjing University (1998)
M.E., Nanjing University (2001)
S.M., Massachusetts Institute of Technology (2003)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2008

Author .....
Department of Electrical Engineering and Computer Science
May 22, 2008

Certified by .....
Karen R. Sollins
Principal Research Scientist, Computer Science and Artificial Intelligence Laboratory
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Agent Organization in the Knowledge Plane

by

Ji Li

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2008, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In designing and building a network like the Internet, we continue to face the problems of scale and distribution. With the dramatic expansion in scale and heterogeneity of the Internet, network management has become an increasingly difficult task. Furthermore, network applications often need to maintain efficient organization among the participants by collecting information from the underlying networks. Such individual information collection activities lead to duplicate efforts and contention for network resources.

The Knowledge Plane (KP) is a new common construct that provides knowledge and expertise to meet the functional, policy and scaling requirements of network management, as well as to create synergy and exploit commonality among many network applications. To achieve these goals, we face many challenging problems, including widely distributed data collection, efficient processing of that data, wide availability of the expertise, etc.

In this thesis, to provide better support for network management and large-scale network applications, I propose a knowledge plane architecture that consists of a network knowledge plane (NetKP) at the network layer, and on top of it, multiple specialized KPs (spec-KPs). The NetKP organizes agents to provide valuable knowledge and facilities about the Internet to the spec-KPs. Each spec-KP is specialized in its own area of interest. In both the NetKP and the spec-KPs, agents are organized into regions based on different sets of constraints. I focus on two key design issues in the NetKP: (1) a region-based architecture for agent organization, in which I design an efficient and non-intrusive organization among regions that combines network topology and a distributed hash table; (2) request and knowledge dissemination, in which I design a robust and efficient broadcast and aggregation mechanism using a tree structure among regions. In the spec-KPs, I build two examples: experiment management on the PlanetLab testbed and distributed intrusion detection on the DETER testbed. The experiment results suggest a common approach driven by the design principles of the Internet and more specialized constraints can derive productive organization for network management and applications.

Thesis Supervisor: Karen R. Sollins
Title: Principal Research Scientist, Computer Science and Artificial Intelligence Laboratory

Acknowledgments

First of all, I would like to sincerely thank my thesis advisor, Dr. Karen R. Sollins. Without her wise guidance and constant support, this dissertation would not be possible and I could not reach this stage of my life. Her vision and insights help me explore broad research areas while focusing on the important problems. She is also always very nice, encouraging, and patient, helping me improve my writing and communication skills. I am extremely fortunate to be able to work with Karen for many years at MIT.

I would also like to thank my thesis committee members, Dr. Arthur Berger, Dr. David Clark, and Professor Barbara Liskov, who have provided insightful suggestions and comments on my dissertation. The thesis is based on Dave's original idea on the knowledge plane. In spite of his busy schedule, Dave is very accessible and spent much time discussing important issues with me. Arthur always encourages and supports me, and his mathematical background significantly improved the rigor of the proofs and the analyses in the thesis. Barbara provided valuable comments from the perspective of distributed systems, and helped me on the writing as well.

Many other researchers also helped me. Jack Brassil, my supervisor at HP Labs, broadened my knowledge and skills in network research, and I had a wonderful internship at Princeton, NJ during the summer of 2005. John Wroclawski, now director of computer networks division at USC ISI, provided many insights on network research during the group meetings. Professor Dina Katabi at MIT and Professor Xiaowei Yang at UC Irvine gave me sincere and valuable suggestions on my research directions, which I really appreciate. Professor Polina Golland kindly allowed me to be her Teaching Assistant for the course *6.041 Probabilistic Systems Analysis & Applied Probability* during the fall of 2005.

It has been my pleasure to have worked with many talented students at the Advanced Network Architecture group: Mike Afergan, Steve Bauer, Rob Beverly, Indraneel Chakraborty, Peyman Faratin, Simson Garfinkel, Richard Hansen, Joanna Kulik, George Lee, Ben Leong, Nishanth Sastry, and many others. I made many friends at MIT: Dah-Yoh Lim, Roger Moh, Jinyang Li, Chang Li, Xue Xiao, Hai Jiang, Fuwan Gan, Jing Zhou, Jianping Fu, and many others, who helped me in many ways and made my life more enjoyable. I would espe-

cially thank Ben Leong, now a professor at National University of Singapore, and Dah-Yoh Lim, a PhD student who is graduating soon. We collaborated on many class and research projects, and I learned a lot from them. I thank Becky Shepardson, Lisa Gaumond and Susan Perez for their administrative work and assistance. I am grateful to Marilyn Pierce, the former EECS graduate administrator, for her advice on various issues.

This work is funded in part under NSF Grant ANIR-0137403, "Regions: A new architectural capability in networking", NSF Grant CCR-0122419, "The Center for Bits and Atoms", NSF Grant ANIR-0626904, "NeTS-FIND: Model-Based Diagnosis in the Knowledge Plane", a gift from the Cisco University Research Program, and a research fund from the Intel Corporation. I thank their generous support.

I am grateful to my advisor at Nanjing University, Guihai Chen, and other professors, Sanglu Lu, Daoxu Chen, and Li Xie. Without their guidance and support, I could not have come to MIT in the first place. My longtime friends from Nanjing University, Mingliang Wei, Tianning Li, Peirong Fu, Jianxin Guo, Wenkai Tan, Ningning Hu, Ming Zhang, Yu Qian, Qiang Wang, and many others, have provided generous help on many aspects of my research and life here.

I have a great family behind me. My parents, brother, and sister-in-law have been always supporting, encouraging, and trusting me. My wife, Haiying Zhu, is always by my side. They deserve my sincerest gratitude.

Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

1.1 Background

The Internet began in the 1960s. At that time, multiple networks coexisted, including ARPAnet, UUCP, etc. TCP/IP was developed to unify these networks so that they could communicate with each other. With the commercialization of the Internet in the late 1980s and early 1990s, the Internet has grown exponentially in its scale and heterogeneity. The number of users has increased from a few in 1960s to more than one billion by 2007; the coverage has expanded from a few universities in the U.S. to almost every country in the world; the devices connected to the Internet have evolved from a few terminals to millions of computers, PDAs, cell phones, robots, and even coffee machines; the applications have increased from email to web, file sharing, online business, gaming and many others. The Internet has become an indispensable component of our society.

Due to its extraordinary success, the Internet has become increasingly global, crossing domains of responsibility, as well as having been pushed deeply into our daily life and very personal environments. It has also evolved from a small, simple, and friendly network to a large and complicated environment where entities have different and sometimes conflicting objectives. Viruses, worms, and spams also leverage the Internet to propagate themselves to their advantage. Today the Internet infrastructure is a federation of more than 15,000 routing domains, each of which is under its own operation and administration. The Border Gateway Protocol (BGP) connects those networks into the Internet.

1.2 New Challenges

Traditional network management has been focusing on managing the elements in the network such as routers and switches. With the increasing density and expansion of the network, the problem space is becoming much larger and more complex, and has clearly expanded into domains of supporting network applications. Therefore, by *network management* we mean a much broader set of problems than the traditional domain of core network management.

For purposes of explanation, we partition the space of network management into managing the network resource and supporting network applications. We will talk about these two topics separately. We further discuss an example, because it has an interesting feature: It essentially happens in the domain of application level management, but it crosses the boundary into the traditional network management as well.

To facilitate the discussion, I use the term "agent" to describe a participant that works together with others to perform a task. Agents function on behalf of users, service providers, etc, and they do a broad range of tasks, ranging from simply collecting packet traces to running sophisticated intrusion detection techniques. An agent can be a local intrusion detector on an end host, a traffic monitor on a gateway, a daemon that provides network topology information, etc. An agent may issue requests to other agents. A "request" is a message that looks for an answer to a problem, or asks for an action to be undertaken. For example, a request may be sent by a local detector to collect intrusion detection status from many other detectors for an aggregate analysis. I will discuss these terms in detail in Chapter 3. The problem we address in this thesis is organizing agents in order to achieve a broader definition of network management.

1.2.1 Network Resource Management

The Internet was originally designed to be a decentralized, multi-administrative structure that combines a simple, transparent network with rich end-system functionality [34, 111]. The core network is simple and transparent in the sense that it only deals with one task: to carry packets without knowing what is in the packets. Rich end-system functionality is

achieved by end hosts at the edge of the Internet, who understand what the goal is and what the applications do in their contexts. This transparency and rich end-system functionality, however, becomes a nightmare for network operators who manage the network at the low level, because they cannot specify the high level goal or even relate the high level goal to the low level operations. This gap leads to a difficulty when the edge recognizes a problem but the network has no idea about what should happen. Traditionally, network analysis, diagnosis, and management have been done manually by a small number of people. As the Internet continues to grow in reach and in density, there are increasing problems with understanding how it works, where it runs into problems, and how to address those problems. As a result, those traditional manual approaches require an increasing number of people to be involved, and the management task itself becomes increasingly complex. Furthermore, some problems are not in the unique domain of network management. The definition of *network management* is expanding. Even within the scope of the network, *network management* refers to a broader spectrum of problems that is not limited to router configuration and other traditional network management tasks.

Let us consider a network diagnosis example. When a web browser is experiencing slow access to a web page, how can the root cause of the problem be identified and fixed, if possible? If network operators or IT support staff are immediately available, they may be called upon. However, this is usually not the case; otherwise, many more support staff will be needed. If a user wants to diagnose the problem by himself, he can first check whether the local host is misconfigured. If not, he needs to check whether the website's IP address is correct, i.e., whether the DNS is working. Then he checks whether the path from the end host to the website is working, and finally whether the web server is overloaded. We can see that considerable knowledge about the network configuration and conditions and real-time measurement are needed for the diagnosis, as the web access requires many interdependent components to work together correctly. Some components are not easy to measure or diagnose from the user side, such as the path condition and the status of the web server. Therefore, more powerful and intelligent mechanisms are needed to provide the necessary knowledge about each component and to automate the diagnosis process.

Actually the problem is even worse, because in general the problem can be caused by

an aggregate behavior of multiple components, as opposed to a single component analyzed above. For example, if every link has a small packet loss rate, the aggregation of the link losses can cause the path to fail intermittently. This fault cannot be traced back to any single device, and requires collecting information from multiple places (links along the path in this example).

Another diagnostic approach is for agents on end hosts to collaborate with each other to confine the scope and the cause of the problem. For example, if many end hosts visiting the web server from different places of the Internet all experience slow browsing, then it might be concluded that the root cause is very likely a problem with the web server itself; if only a set of neighboring hosts are having the same problem, then it is likely a local area network problem. Therefore, it is very valuable to have a mechanism that helps agents discover each other and collaborate together based on network scope/topology.

Part of the problem space that has received lots of attention so far is single source fault diagnosis. I am more interested in the other side of such problems. In this thesis, I study problems that involve multiple components simultaneously, and focus on the collaboration among many agents in order to solve problems more effectively.

1.2.2 Network Application Support

An orthogonal issue to network management is the support needed by new network applications. Those applications are widely distributed across the Internet, and often maintain their own connectivity graphs among their participants. Examples include overlay networks, content distribution networks, end system multicast, peer-to-peer networks, publish/subscribe systems, etc [16, 33, 49, 67, 102]. Routing overlays build their own routing graphs to route around congested paths with comparable or even better performance [16]; end-system multicast constructs the application-layer multicast tree to transmit video efficiently [33]; nodes in peer-to-peer networks probe each other to find nearby neighbors to improve lookup performance [49]. Currently each application builds and maintains its own connectivity graph by probing latency, available bandwidth, loss rate or other metrics actively between hosts, which often incurs significant cost in the network as redundant

operations are performed by them individually. Due to limited network resources, such redundant operations cause contentions for resources, instability of the network traffic, and many other potential problems [10].

To summarize, the new features common in many network applications are as follows:

- The systems are widely distributed in the Internet, and are formed as a set of entities that collaborate to achieve certain functionality under some constraints;
- They need information from the network layer to organize themselves efficiently, and there is a commonality in the structure and nature of the pattern that it is both useful and possible to abstract out;
- They involve a variety of expertise and reasoning in order to handle widely distributed, incomplete, possibly contradictory and inaccurate information.

1.2.3 A Motivating Example

For motivation, let us consider a specific example: intrusion detection. Please note that this is just an example used to demonstrate the challenges we will address in this thesis. These challenges are common in many other problems in today's Internet. The same example will be carried out through the following chapters, and I will discuss it in depth in Chapter 7.

In this example, we focus on zero-day and slow-scanning worms. Such worms are especially difficult to detect due to two features. First, the worms are completely new, which means no existing signatures or behaviors are known. Thus, many existing intrusion detection systems that depend on signatures do not work. Second, unlike traditional worms that propagate aggressively in the Internet, slow-scanning worms propagate slowly to hide themselves from detection. Although a gateway is a natural place to inspect aggregated traffic, without knowing the signature beforehand, it is hard to observe strong traffic patterns from those slow-scanning worms. To that end, since the behaviors of zero-day worms are not known a priori, the best location for initial attention is the local host itself, because only the end node can possibly know whether slightly anomalous traffic is a potential attack or only a new application, in the context of local behaviors. However, at the local node,

one loses the aggregation effect of repeated or simultaneous low level anomalies. In addition, it is difficult to make all local detectors strong enough because they see only a small percentage of the global traffic and cannot devote many resources to detection. Therefore, there is a need for those weak local detectors to collaborate efficiently, so that the system can aggregate the results of weak local detectors to get a broader perspective and to find stronger signals. This idea is appearing increasingly in other places as well [59, 60].

Based on the above analysis, we face the following challenges. These challenges are common in many network management and network applications in today's Internet. First, a more effective detection approach is needed. As discussed above, detectors at the gateways cannot detect such worms effectively, nor can an individual local detector, so we propose to aggregate information from many local detectors and conduct an aggregate analysis. To generalize this idea, let us think about the agents in the Internet and the knowledge they have. There are many detectors with various capabilities, and many different kinds of useful knowledge in the network. For example, some agents may have the expertise to analyze aggregate data, and some may have knowledge about the network topology and configuration information that can be used to cluster the local detectors together. Examples of questions to be analyzed are: How can those local detectors or agents with similar interests find each other in the first place? How can the detectors find the necessary knowledge and techniques to conduct the aggregate analysis? To address these questions, we discuss the agent organizing criteria and principles in Chapter 3. We further propose a mechanism that supports an agent to issue a broadcast request to other agents so that those with similar interests can find each other.

Second, the overhead of the detection approach must be low. A good approach should not overburden the network. We want to minimize message passing and avoid flooding schemes that generate excessive redundant messages. One way to do so is to organize detectors based on their network proximity. For instance, enterprise networks are reflected in topological neighborhoods and we can cluster detectors in the same enterprise networks together, and detectors outside the enterprise networks may be organized based on the Autonomous System boundaries. In this way, most communication happens among nearby nodes. Such network knowledge is useful in many other areas, such as server selection,

application layer multicast, and file sharing [49, 58]. Therefore, we propose to build a new infrastructure to provide knowledge about the Internet, as discussed in Chapter 3. We further discuss how to achieve efficiency and non-intrusiveness in the organization of a large number of agents in Chapter 4.

Third, the approach should not create a bottleneck in the network or cause huge traffic explosion at a single point. For example, if we construct a tree structure to aggregate information and each local detector is a node in the tree, we want to ensure that there is good load-balancing, in the sense that no node in the tree should be responsible for forwarding a disproportionate amount of network traffic or have a large number of child nodes to take care of. We present a broadcast and aggregation mechanism based on a balanced tree structure to propagate requests and knowledge in Chapter 5.

Finally, if detectors belonging to multiple parties are involved, very often local control is needed to determine what information to be exposed, as there may be various kinds of constraints, such as security, policy and economics imposed by those parties. For example, an agent may allow the complete packet trace to be shared with others, while another may only allow the sharing of packet headers or even just report whether it detects an attack. We discuss this issue in Chapter 7 to ensure those policies in an effective way.

1.3 The Knowledge Plane

The work presented here falls into the paradigm of the knowledge plane. To make the network more intelligent, *the knowledge plane* was proposed by Clark et al. in [35]. In network architecture, we recognize two architectural divisions: a data plane over which data is transported, and a control plane that manages the data plane. The knowledge plane (KP) is a new higher level construct in network architecture, contrasting with the data and control planes. Its purpose is to provide knowledge and expertise to enable the network to be self-monitoring, self-analyzing, self-diagnosing, and self-maintaining. At an abstract level, the knowledge plane gathers observations, constraints and assertions, and applies reasoning to these to generate observations and responses. At the physical level, it runs on hosts and servers within the network on which knowledge is stored. The KP is a loosely

coupled distributed system of global scope. The KP brings up a number of challenging problems, such as knowledge representation and dissemination, incorporation of AI and cognitive techniques, conflict resolution, trust and security, how to design a distributed knowledge base for the networks, how to incorporate into the framework pre-existing sets of specialized data and tools for network management, etc.

We believe that the knowledge plane is an appropriate construct to address the new challenges from network management and network applications. Based on the types of knowledge at the network layer and the upper layer, as well as the organization suitable for those layers, I propose an architecture that consists of the following components:

1. *A network knowledge plane (NetKP).* The NetKP is an application-independent mechanism at the network layer that provides knowledge about network topology, conditions, policies, etc. The NetKP supports network management and applications.

2. *Specialized KPs (Spec-KPs).* The spec-KPs are application-specific, and specialize in various areas, to achieve certain functionality of network management or applications, under a set of constraints.

To design and build the NetKP and the spec-KPs, we identify the following design requirements: scalability, to address the size and scope of the Internet; efficiency, to provide responsiveness to requests; robustness, to enable the KP to continue to function as best possible, even under incorrect or incomplete behavior; non-intrusiveness, to keep the KP from impinging significantly on the resource usage intended for the customers of the network. Spec-KPs also need to satisfy additional constraints from their own areas of interest, and have local control to support local networks and resources in their needs for privacy and other forms of local control, while enabling them to cooperate for mutual benefit in more effective network management.

1.4 Thesis Contributions

In this thesis, I make the following contributions.

First, I propose a new architecture for network management and applications based on the concept of the knowledge plane. The architecture consists of a network knowledge plane and, on top of it, multiple specialized KPs. The NetKP provides network knowledge and facilities to help construct the spec-KPs. Each spec-KP is specialized in its own area of interest under functionality, policy and other constraints.

Second, I analyze the criteria and principles to organize the NetKP and the spec-KPs, and propose a region-based agent organization to support those criteria and to operate within the principals.

Third, I address two important issues in the NetKP. One is the cross-region organization, in which I design a distributed hash table that leverages network topology knowledge to achieve efficiency and non-intrusiveness. The other is a broadcast and aggregation mechanism among regions, in which design a robust and efficient tree construction and maintenance protocol using a novel idea of the parent function.

Fourth, I conduct two case studies on the spec-KPs: experiment management on testbeds and distributed intrusion detection. In the first case study, I study how to facilitate distributed experiment management on PlanetLab, specifically on how different kinds of knowledge are maintained and propagated to resolve the node selection problem during the experiment setup. In the second case study, I design a knowledge-based framework for collaboration between different intrusion detection systems, and implement a distributed intrusion detection system on the DETER testbed.

Fifth, during the research I also developed several insights that are not limited to the knowledge plane and can be applied to many other areas in the large-scale distributed systems.

- Latency approximation. I use the number of Autonomous System (AS) hops as an approximation of the end-to-end latency, and take the AS heterogeneity into consideration. The proposed hybrid proximity neighbor selection algorithm achieves the lookup performance comparable to proximity neighbor selection while significantly reducing the probing traffic.

- Parent function. The parent function provides a flexible way to construct and main-

tain a tree structure, as the tree properties are tunable using the parameters in the parent function. The parent function I proposed forms a balanced tree that is robust to node churn and can be optimized according to network topology. It also allows an efficient construction of multiple interior-node-disjoint trees, thus alleviating single points of failure.

- Dependency-based intrusion detection. To detect zero-day and slow-scanning worms, I design an approach that allows for partitioning along policy boundaries and that applies different detection techniques within and across regions to capture the dependency among end hosts.

1.5 Roadmap of the Dissertation

The dissertation is organized as follows. Chapter 2 discusses research work in several related areas, including network architecture, overlay network organization, and our previous research on regions. Chapter 3 presents the system architecture and the organizing principles of the NetKP and the spec-KPs. Chapter 4 presents a cross-region organization in the NetKP that combines network topology knowledge and a distributed hash table. Chapter 5 designs an aggregation and broadcast mechanism for request and knowledge dissemination among regions in the NetKP. Chapter 6 presents a case study on experiment management on PlanetLab, to demonstrate how different types of knowledge are maintained and propagated based on their features. Chapter 7 proposes an intrusion detection framework, and presents a distributed intrusion detection system on the DETER testbed. Chapter 8 concludes the dissertation and discusses the future work.

Chapter 2

Related Work

There are many categories of related work to the knowledge plane, as many issues are involved in this work. In this chapter, I present an overview of network architecture, overlay networks, broadcast and aggregation mechanisms. More specific related work to each chapter will be discussed in those chapters.

2.1 Network Architecture and Management

In the original Internet architecture, distributed management of networks under different administrations was not given high priority, as other goals, such as multiplexed utilization of existing interconnected network, survivability, and support for multiple types of service [34]. However, as the Internet increases its size and heterogeneity, and becomes an indispensable part of the infrastructure, network management has become a hard problem. That is why the knowledge plane is proposed to make the network self-manageable [35].

New architectures have been proposed to meet the challenges in network management. Greenberg et al. proposed a clean-slate approach to network control and management, the 4D architecture [51]. 4D employs a centralized top-down method guided by three principles: Specify network-level objectives for performance, reliability, and policy; Collect timely and accurate network-wide views of topology, traffic, and events; Use direct control on switches instead of hardwired logic. 4D decomposes the functions of network control into four planes: a decision plane that is responsible for creating a network configuration; a

dissemination plane that gathers information about network state to the decision plane, and distributes decision plane output to routers; a discovery plane that enables devices to discover their directly connected neighbors; and a data plane for forwarding network traffic. 4D separates the decision making and the placement of the information and the execution, and suggests a centralized method for the decision making. We concur with the value of the separation of the functionalities, and focus in this work on how the expertise is organized efficiently in a distributed way. 4D is proposed for network control and management within an administration, while our work aims at the global Internet where thousands of different administrations coexist. As a result, our focus is more on how agents are organized into regions and how regions collaborate to resolve network problems and facilitate network applications.

Feamster takes a different approach towards network management, specifically on Internet routing [45]. Instead of fixing the problems afterwards, he proposes to define a specification for correct behavior of a routing protocol in terms of path visibility, route validity, and safety. Furthermore, he develops techniques to check whether a routing protocol satisfies the correctness specification within a network and whether the interactions of multiple networks violate the correctness.

iPlane [75] is a common infrastructure for collecting network data. Compared with many of its predecessors, iPlane provides a richer set of performance characteristics using several clustering techniques. iPlane and many other research efforts, including this work, realize that there is a set of common functions that should be provided to network management and applications, instead of letting each individual application implements its own functions. iPlane only focuses on network performance information, while the work in this thesis provides a broader set of functions. iPlane employs a centralized way to manage and disseminate the knowledge, which may not scale to a large number of participants. In this thesis, we choose a distributed model to organize agents and resolve requests, as not all queries go to the same place and the set of resource to examine may not overlap with each other.

Lee proposes CAPRI, a common architecture for distributed probabilistic fault diagnosis on the Internet [69]. CAPRI addresses several challenges including the extensible

representation and communication of diagnostic information, the description of diagnostic agent capabilities, and efficient distributed inference. It is an open question where the knowledge comes from. This thesis does not address this rendezvous problem, and rather focuses on scalable and efficient agent organization for request and knowledge dissemination.

Steinder et al. discuss various techniques and algorithms on Internet fault diagnosis in [121]. The diagnosis algorithms come from many different fields including decision trees, neural networks, graph theory, etc. They also propose a hierarchical partitioning of the problem space [120]. In that approach, a diagnose manager divide the diagnosis into multiple domains, and each domain performs local diagnosis separately. Then local diagnosis results are collected and analyzed by the manager to draw a global conclusion. In a similar spirit but more broadly, I propose to diagnosis network failures from the end-to-end point of view, so our approach crosses multiple network layers and services, not just about routing.

Ballani et al. propose Complexity Oblivious Network Management (CONMan), a network architecture in which the management interface of data-plane protocols includes minimal protocol-specific information [19]. CONMan restricts the operational complexity of protocols to their implementation and allows the management plane to achieve high level policies in a structured fashion. The *Network Managers (NMs)* are similar to the *agents* in our work that manage network management. So far CONMan has been focusing on a single NM and its modules. Our work in this thesis tries to solve the scalability problem when we have many (maybe millions) of agents distributed in the Internet, instead of the implementation of specific agents. Furthermore, CONMan does not deal with non-locality, and is limited to a local domain. In contrast, we address in this work how agents organize themselves into regions, and how agents collaborate with each other within a region and among regions.

2.2 Agent Organization and Overlay Networks

A research effort closely related to the NetKP is the routing underlays [82, 83]. Routing underlays use network topology information to eliminate redundant virtual links in overlays to achieve scalability. Both routing underlays and the NetKP try to expose network topology knowledge to the applications. However, there are several differences. First, the NetKP is designed to be part of the KP under a general framework, and is to be extended to accommodate other kinds of knowledge. Second, it provides more knowledge than routing information. Third, it aims to help many applications in different ways, not only routing overlays.

Many peer-to-peer networks provide a robust and scalable organization among dynamic nodes. Based on how the nodes are connected to each other, P2P networks are classified into two kinds: unstructured, such as Napster [85], Gnutella [49], KaZaA [66], Freenet [36], and structured, such as CAN [104], Chord [122], Tapestry [133], or Pastry [109]. In unstructured P2P networks, nodes are connected somewhat arbitrarily, often without a strict or well-defined structure. Structured P2P networks, especially distributed hash tables (DHTs), employ a globally consistent protocol to ensure that any node can efficiently route a search to its peers. A DHT is a distributed resolution mechanism for P2P systems that manages the distribution of data among a changing set of nodes by mapping keys to nodes. DHTs allow member nodes to efficiently locate stored resources by name without using centralized servers. A large number of DHTs, such as CAN Chord, Tapestry and Pastry, have been proposed. These systems are expected to eventually become the fundamental components of large scale distributed applications in the near future, and would therefore require an aggregation/broadcast functionality. In this work, I use a combination of the DHT and network topology in our region organization, as described in Chapter 4.

Another related work is large-scale measurement and monitoring infrastructures, which usually manage a large number of monitoring hosts. Many measurement and monitoring infrastructures have been proposed and built so far [57, 8, 130]. TAMI [57] is a measurement infrastructure that is both topology-aware and supports various scheduling mechanisms. But the topologies in TAMI are source and sink trees only, mainly for bandwidth

measurement purposes. Those infrastructures only support basic operations among their members, mostly monitoring and probing. Our mechanism makes use of network topology such as autonomous systems information. Agents are organized based on their topological locations. In contrast with the simple behaviors in the previous measurement and monitoring infrastructures, agents in the knowledge plane can collaborate to perform intelligent reasoning. The prototype mechanism in this thesis is aimed to be a step forward from the simple infrastructure and distributed query processing toward the KP. KP will need to build on these and new measurement monitoring capabilities as underpinnings as well as possibly supporting the management and organization of such distributed tools. NetProfiler [89] uses a peer-to-peer organization to facilitate end-to-end fault diagnosis using aggregation.

Semantic overlay networks [124, 37, 26] are similar to the issues we have in organizing agents in the network applications in that both focus on the dissemination and sharing of complicated knowledge that requires semantic understanding. Interest-based overlay networks such as [118] are similar to this work in that peers prefer to connect with others with similar interests. In that work, a node caches the information about other peers which provided useful results to recent queries, as those peers have shown similar interests, and are likely to provide good results for the future queries.

2.3 Propagation and Aggregation

Publish/subscribe systems address problems similar to ours [67, 102]. There are three kinds of pub/sub systems: unicast, single-identifier multicast and content-based multicast. Unicast systems transmit notifications directly to subscribers over the Internet's existing mechanisms. Single-identifier multicast systems send messages to discrete message channels to which customers with identical interests subscribe. The most popular approach is content-based multicast systems, which forward messages based on the text content of the messages. The problem in the KP is more closely related to content-based multicast than the other two in that in our problem, requests and knowledge need to match each other based on content. However, our problem is more complicated because there are many heterogeneous knowledge sources with different interests and capacities.

Search has been an important component in many networked systems, especially in peer-to-peer systems. For example, Distributed Hash Tables use hashed IDs to find specific files [122, 104, 133]. Gnutella [49] uses scoped broadcast of key words to search for contents, which is similar to our propagation problem. However, in the KP, more complete representations of requests and knowledge, and more sophisticated search functions are needed, such as range search, fuzzy pattern match, etc.

Content routing networks support name-based routing in the Internet [52, 9]. In [52], routers need to support name-based routing which requires name-based aggregation. However, the current name aggregation mechanism is not scalable. The KP does not require the routing infrastructure changes, but we will need similar aggregation functionality to aggregate requests among agents in both the network layer and the application layer. Intentional naming system (INS) [9] integrates name resolution and message delivery by constructing a spanning-tree overlay for anycast and multicast. INS employs "late binding", where the binding between the name and the network location is made at message delivery time rather than at the request resolution time, to handle mobile and dynamic situations. However, INS is designed for intra-domain deployment and does not scale to the Internet scope, and the "late binding" is expensive for content distribution.

Directed diffusion is an important data dissemination paradigm in sensor networks [61, 73]. In directed diffusion, a sensing task is disseminated throughout the sensor network as an interest for named data, and this process sets up gradients to draw events that match the interest. Then events start flowing towards the originators of the interest. We have a similar goal in the knowledge plane: how agents organize themselves to resolve requests. However, sensor networks are different from the Internet in that sensor networks are smaller in the scale and simpler in the structure.

The semantic web is a task force aimed to make web pages understandable by computers, so that websites can be searched in a standardized way. The potential benefits are that computers can harness the enormous network of information and services on the Web. The semantic web uses the descriptive technologies Resource Description Framework (RDF) and Web Ontology Language (OWL), and the data-centric, customizable Extensible Markup Language (XML), to address the machine-readability problem. These existing

techniques help us with the knowledge representation problem, but they cannot solve the propagation problem.

2.4 Region Research

In my previous research, the Region Project [98, 117] is related to this thesis. In that work, I designed and implemented a region mechanism for peer-to-peer systems to improve lookup and replication performance using Autonomous System information [71]. The *regions* in that work can be viewed as a simplified prototype of the KP, since regions provide the applications with the underlying network topology information, but lacking in the sophisticated support that the KP provides.

Similar ideas have been proposed in other research fields. For example, in sensor networks, there have been research on microprogramming at "region" level [86]. In [86], a region represents a collection of signals, and is defined in terms of broadcast hops. Our regions exist at both the network level and application level, and can be defined in different criteria. Whitehouse et al. proposes the concept of "Hood" in [127]. In Hood, a neighborhood is fundamentally a local construction at each node, in contrast to the regions in our work where membership is shared and manipulated among all the nodes in the group.

2.5 Summary

I have discussed several research areas that are related to the knowledge plane. In some later chapters I will discuss related work specific to the research problems addressed by those chapters in more detail.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

System Design

To address the challenges in network management and network applications, I propose to construct an application-independent network knowledge plane at the network layer, and on top of it, to build the application-dependent spec-KPs. In this chapter, I present the system architecture, and describe each component in the architecture. Section 3.1 presents an overview of the system architecture, and defines several important concepts. Section 3.3 describes the network knowledge plane in detail. Section 3.4 briefly discusses how to organize the spec-KPs using network knowledge and area-specific knowledge and constraints. Section 3.5 summarizes this chapter.

3.1 System Architecture

3.1.1 Overview

The knowledge plane provides us a unified approach that encourages edge involvement from a global prospective. Based on the idea of the knowledge plane, I propose to build a network knowledge plane (NetKP) at the network layer and on top of it, multiple specialized KPs (spec-KPs), each specialized for one network management task or a network application. Their relationship is demonstrated in Figure 3-1. Both the NetKP and the spec-KPs are composed of agents, and agents may be in a single KP or multiple KPs at the same time. In each KP, agents are organized into regions for scalability and efficiency.

We propose the division between the NetKP and the spec-KPs for the following reasons. First, the NetKP is at the bottom level of the Internet, and is not dependent on any other layers. In order to do that, we base its organization on the underlying, pre-existing network structure, specifically network topology. On the other hand, because the spec-KPs are built on top of the existing structure, we allow for more diversified organization based on high level requirements, more sophisticated functionality and constraints.

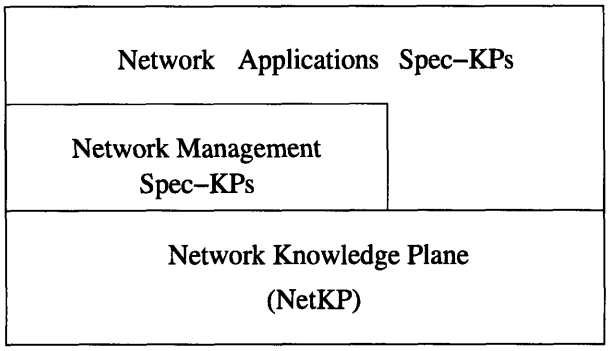


Figure 3-1: The system architecture. The underlying is the NetKP that provides application-independent network knowledge and facilities. On top of it, there are two parts: network management spec-KPs and network application spec-KPs. Network management spec-KPs is needed to support some network application spec-KPs.

The NetKP is designed to provide an increasing set of capabilities to network management and applications in a scalable and efficient way. The knowledge provided in the NetKP includes network topology, network conditions (latency, bandwidth, etc), policy information, etc. The NetKP also provides facilities to help agents in the spec-KPs to discover each other, organize themselves, and communicate in different ways (unicast, broadcast, etc). For example, the performance downgrade diagnosis in Chapter 1 can use the knowledge provided by the NetKP to find out network conditions, find others who are experiencing similar problems, and collaborate to figure out the root cause. To build such an infrastructure, I will address two problems: one is the agent organization using network knowledge, and the other is a broadcast and aggregation mechanism for knowledge dissemination. Those two topics are addressed in the next two chapters.

On top of the NetKP, there are multiple specialized KPs for network management and applications, using the knowledge and facilities provided by the NetKP. The organizations of the spec-KPs are constructed and maintained using the knowledge provided by

the NetKP, knowledge in their specific areas, and other constraints. Each spec-KP specializes in one network management task or application. For example, agents interested in the fault diagnosis form a spec-KP that helps diagnose network failures or performance downgrade, while agents interested in security form an intrusion detection spec-KP.

Note that we treat network management as the spec-KPs on top of the NetKP, instead of part of the NetKP, for the following reasons. The NetKP only provides basic knowledge and primitives about the network conditions, and does not perform sophisticated reasoning. Network management requires the understanding of high-level objectives and constraints that cannot be easily captured by the NetKP. Therefore, we treat network management as separate spec-KPs on top of the NetKP.

3.1.2 Agents

An agent-based model is a conceptual paradigm for analyzing problems and for designing systems, for dealing with complexity, distribution, and interactivity, while providing a new perspective on computing and intelligence [18, 24, 74]. We choose an agent-based model for the following reasons. First, entities in our system are autonomous and represent different parties. Some represent end users, some represent network operators, some represent network application developers, and each has its own goal. Second, entities collect and manage knowledge, learn, and reason, based on their own capabilities. Third, entities interact with each other to achieve their goals. Therefore, an agent-based model fits our needs.

Both the NetKP and the spec-KPs are composed of agents. In this thesis, *an agent* is defined as an autonomous entity in the Internet that manages knowledge, performs reasoning, and interacts with other entities in the NetKP or the spec-KPs. Agents are responsible for collecting, storing, sharing and distributing requests and knowledge. A *request* is a message that looks for an answer to a problem, or asks for an action to be undertaken. Agents also manage a distributed knowledge base. The knowledge base is distributed among agents since the network knowledge is distributed and managed by different parties in the Internet. The NetKP is composed of agents at the network layer, and the spec-KPs are composed of

agents in their specific areas of interest.

I believe that there are two aspects of an agent; each agent provides at least one of these. One is request propagation, and the other is request resolution. There are two reasons to separate these two aspects. First, propagation is a common task in the KP, and they are similar in all the agents. Second, request resolution often requires reasoning and inference techniques, which are different in different agents. Depending on their capacity, willingness, and availability, agents may have one or both of the components. These two separable aspects reflect the fact that different agents may have different roles and capabilities in the KP: some agents are responsible for receiving requests and responding to them, and some manage the knowledge, and some do both.

Agents are deployed by different parties, such as end users, ISPs, application developers. Agents have access to different kinds of knowledge. For instance, in the intrusion detection example in Chapter 1, an end user may run an agent on his computer as a weak local detector, while the corporate can deploy agents on dedicated machines who collect and analyze the reports from many such local detectors and that have access to the gateway information as well.

Some agents are stable, while others may join and leave frequently. For example, an agent for fault diagnosis may only join the system and collaborate with others when it discovers some problems, while an agent for intrusion detection in a corporate network may run for a long time keeping monitoring the network and exchanging information with other agents. Therefore, a robust organization among agents is needed.

3.1.3 Region

Due to their large number, agents need a scalable and efficient organization. I follow the divide-and-conquer strategy by dividing the agents into regions. A region is an entity that encapsulates and implements scoping, subdividing, and crossing boundaries of sets of entities that share common features or constraints [117]. A region is a new design element in the network architecture for large scale, wide distribution and heterogeneous networks. In this work, I use the region as the building block of the NetKP and the spec-KPs, for the fol-

lowing reasons. First, due to the large scale of the knowledge plane, we need to introduce some organizing structure to achieve scalability and efficiency, and a region is a natural structure that fits our needs. Second, as a general and flexible networking mechanism, a region or set of regions can be tuned to match the Internet structure which consists of tens of thousands of different administrative domains, or match the inherent structures of the spec-KPs.

Agents are organized into regions for scalability and efficiency. Agents are grouped into different regions according to certain criteria, such as network proximity, interest, etc, as we will discuss in the following sections. In the NetKP, we organize agents into regions based on network topology and other network-level constraints. In the spec-KPs, a region is defined by a bunch of constraints, both application-independent and application-dependent. In the intrusion detection example, due to security and policy concerns, agents can collaborate closely with only those in the same organization, so the regions are formed based on policy boundaries, such as enterprise networks. We will discuss the constraints in both the NetKP and the spec-KPs in detail later.

Agent Types

According to their roles in the regions, we classify agents into three kinds in our system: member agents, regional leaders, and global agents. This classification applied to agents both in the NetKP and in the spec-KPs.

1. *Member agent.* A member agent has three tasks. First, it maintains the connection with neighboring agents and the leader in its region. Second, it resolves requests using its knowledge. Third, it acts as a proxy to resolve the requests from local users. Some agents may specialize in some areas, such as network topology or geography knowledge service.

2. *Regional leader.* Each region has a leader. The regional leader maintains a list of all the agents in its region. It helps a new agent join its region by providing information about existing agents. It also maintains connections with other regional leaders. A

leader is a hub that connects the member agents in the local region to the outside world.

3. *Global agent.* Global agents are those who provide global services to all the agents. Currently there is only one such kind of global agent: the global directory agent. The global directory agent maintains a list of existing regions, and helps a new agent find appropriate regions to join. It also helps to detect inconsistency caused by agent failures or network partitions. Note that global agents are not required in agent organization, and can be replaced by any other bootstrap mechanisms. All the other functions can be provided by leaders themselves.

Agents can also be classified into three categories based on their owners and privileges. One is the authoritative agents deployed by ISPs, institutions and organizations in their own networks. They may have (limited) access to sensitive information, such as the BGP information at the border gateways, and may enforce certain policy when exposing this knowledge. Another is the agents residing on end hosts. These agents are proxies for the users, and collaborate with each other to resolve requests. A third category from the perspective of a region is the agents outside the region, both peers and other kinds of agents. In this work, we deploy multiple agents with BGP data or geographic data to act as authoritative agents.

In the intrusion detection example, when a user sends a request to a regional leader (regional detector) about whether there are any intrusion attempts, the regional leader needs to collect knowledge from local agents (local detectors), so it sends new requests to many local detectors in its region in the intrusion detection spec-KP. The local detectors in the spec-KP use packet traces to make local decisions. Those packet traces are collected by local agents in the NetKP. The regional detector in the spec-KP makes a regional decision using the information collected from the local detectors, thus creating new knowledge. Note that the regional detector performs both request propagation and resolution functions in this case.

Philosophical Aside

In designing such a system, for many important issues there is more than one design choice. We take particular positions on that spectrum of those designs, and do not guarantee that they are the best ones, but they solve our problems in this work. Key choices are:

1. At least one agent is needed for a region to exist. In this work, regions are formed in a bottom-up way, which is appropriate as the regions are used to manage agents. There are other models in which a region can be empty. For example, we can create a region first, and then put agents or assign members into it. As another example, if regions are defined by the distance to some landmarks, then regions exist whether or not there are agents in them.

2. Agents are static. Note that I assume static agents instead of mobile agents in this work, and communication between agents is through message passing. In the future we may evaluate the need for mobile agents. Depending on the definition of the regions, mobile agents can introduce much complexity into the region formation. For example, if a region is defined by the distance from the current leader, then the membership changes as the leader moves.

3.1.4 Knowledge

Knowledge as a term is more common in the AI community than in the networking. In this thesis, knowledge refers to any useful information in the Internet, including the information about individual objects in the network, and the relationships between objects. There are various kinds of knowledge in the Internet. There are two ways to categorize knowledge. First, based on the subject matter of the knowledge, we can classify knowledge into two kinds: network knowledge at the network layer, and specific knowledge in different areas at the application layer. Second, based on the form of the knowledge, we can classify it into facts and relationships. Sources of knowledge include human beings, measurements and monitoring, and inference, reasoning and learning. A specification of a knowledge domain is usually defined in some ontology languages, such as XML, RDP or OWL [53, 126].

Due to the distributed and heterogeneous nature of the knowledge in the networks and its large amount, we need a distributed knowledge base. I will address issues including knowledge collection and distribution in the following sections, but note that the knowledge base is not the focus of this thesis.

Knowledge Credibility

Credibility is an important issue in the knowledge plane, as agents who provide the knowledge come from different organizations and each has its own interest. Agents must be prepared to deal with knowledge with different credibility. We discuss briefly several problems related to credibility below, including completeness, timeliness, consistency, and correctness.

1. Completeness. It is always better to have complete knowledge, but sometimes it is not possible. Agents may need to deal with incomplete knowledge. For example, an agent may need to figure out the latency between two hosts, but it is only able to get the latency of several links along that path. In the intrusion detection example that we will discuss in detail in Chapter 7, regional leaders are designed to work with incomplete and even contradictory knowledge using statistical learning techniques.

2. Timeliness. Some knowledge is time sensitive, and expires very soon. For example, an agent may need a prediction on the condition of a path in an hour. If the answer comes after an hour, it may not be useful any more.

3. Consistency. Even if all agents are honest, we still need to consider the knowledge consistency from different agents. An agent may receive different answers from different agents. The differences may be due to many reasons. For example, a multi-homed agent may find multiple AS paths between itself and a remote host, and it has to be able to tell which one is what it wants (maybe both). As another example, an agent is likely to receive different answers from different agents about the latency between two hosts, as agents at different locations may use different methods to obtain the latency. In the intrusion detection example, because local agents see different

traffic, their responses may appear to be inconsistent with each other, and we must be able to operate in such an environment.

4. Integrity. An agent needs to figure out whether the knowledge it receives is corrupted or not.

Accordingly, we face challenges such as the provider's intention, scaling, efficiency, etc. An ISP may be reluctant to admit failures within their own networks. As another example of a desire to hide the local state, consider agents residing on end hosts. An agent that represents an end user may want to hide the fact that worms originate from his machine due to his negligence. Furthermore, the amount of knowledge is huge, and how can we find what we need in a scalable and efficient way?

In this prototype network knowledge plane, we set aside malicious intentions. That is, agents may provide incomplete knowledge, but we assume they will not provide false knowledge intentionally. As future work, a trust model is needed that considers both authentication and reputation. In Chapter 7 we will consider trust and private information retrieval in collaborative intrusion detection.

3.2 Key Design Issues

To build the NetKP and the spec-KPs, which are distributed systems of global scale like the knowledge plane, the following requirements must be met:

1. *Scalability*. The number of agents is large and they are distributed all over the Internet, so is the knowledge. We need to organize the agents in a scalable way to support request and knowledge dissemination.

2. *Efficiency*. The NetKP and the spec-KPs are supposed to respond quickly to requests, and distribute knowledge to where it is needed efficiently. Furthermore, in many cases knowledge, such as the available bandwidth of a path, may become outdated very soon. Therefore, we need efficient knowledge collection and dissemination mechanisms.

3. *Robustness*. The knowledge plane functionalities are often most needed when the network is not working correctly. Therefore, agent organization must be robust to network failures, partition, and instability.

4. *Non-intrusiveness*. The knowledge plane needs to collect, share and distribute large amounts of knowledge, without adding too much burden to the network. Therefore, a non-intrusive propagation mechanism is needed for knowledge and requests so as to reduce network traffic overhead.

There are other properties we need to consider. We should design the KP for trustworthiness, longevity, and heterogeneity, but they are not the focus of this thesis.

In both the NetKP and the spec-KPs, there are three tasks in common: (1) agent organization; (2) request and knowledge dissemination; (3) knowledge management. The first task, agent organization, refers to how agents discover each other and organize themselves together. The second task, request and knowledge dissemination, deals with how to propagate requests so that they can be resolved quickly and efficiently and how to disseminate knowledge so that agents interested in that knowledge can receive it in a timely fashion. The first two tasks are tightly related to each other, because agent organization largely determines how requests and knowledge can be propagated. The third task, knowledge management, addresses the question of how agents manage their local knowledge and learned knowledge. For example, agents may maintain a distributed knowledge base. The first two tasks are the focus of this thesis.

As discussed earlier, I propose a region-based structure to organize the NetKP and the spec-KPs. The concept of a region is the central organizing components in this work [117]. Agents are organized into regions to achieve scalability and efficiency. In the NetKP, regions are constructed following network topology, such as Autonomous Systems (ASes) and corporate network boundaries. For efficiency, agents in a large Autonomous System may be divided into several regions, while agents in nearby small ASes may be in the same region. In the spec-KPs, agents with similar interests form regions based on both network topology and area-specific knowledge and constraints.

3.3 Network Knowledge Plane

The NetKP is an application-independent distributed system that provides network knowledge to help construct the spec-KPs for network management and applications. For example, the intrusion detection example needs packet traces, but the agents in this spec-KP do not collect the traces themselves; instead, they get the data from the NetKP. Initially, this information will be the basic information, measurements and other kinds of network specific information, both reasonably static and often quite dynamic. The NetKP may be seeded with basic information, but quickly it will also be extended with new, more accurate, more complete, or otherwise more extended information.

In the following, I first describe the functions and interface the NetKP provides, then discuss the agent organization that follows the Internet topology, request propagation and resolution, and knowledge collection in the NetKP.

3.3.1 Network Knowledge and Functions

We divide network knowledge into three categories: configuration, operation, and control knowledge. Network knowledge is provided by the agents in the NetKP to the agents in the spec-KPs. A set of functions is defined in the following accordingly. Note that this is a set of examples, and more will be added as needed.

1. Configuration knowledge. This kind of knowledge includes network topology, geographic location, etc, which are configured to make the network work. Such knowledge is stable, and changes infrequently. The following functions are defined to provide network topology knowledge. They correspond to the requests that an agent receives from others in the NetKP or in the spec-KPs.

* *getASN(IP)*. Given an IP address *IP*, return the AS number to which the IP address belongs. This tells the topological location of an agent.

* *getASPath(AS1, AS2)*. Given two AS numbers *AS1* and *AS2*, return the AS path from *AS1* to *AS2*. This gives a measure of the topological distance between two ASes.

* *getLocation(IP, [metric])*. Given an IP address *IP*, return its geographic location in the form of a *metric*. Currently we use city and state for this metric.

2. Operation knowledge. This refers to the network conditions or the realized network performance, which includes latency, available bandwidth, loss rate, etc. Such knowledge usually changes frequently with time, and needs to be measured at runtime unless a recent measurement is cached; in contrast, configuration knowledge is stable, and can be easily replicated and stored at multiple locations. The following examples are defined below:

* *getLatency(IP1, IP2, [time])*. Given two IP addresses *IP1* and *IP2*, return the latency from *IP1* to *IP2* at *time*, and *time* is an optional parameter.

* *getBandwidth(IP1, IP2, [time])*. Given two IP addresses, return the bandwidth from *IP1* to *IP2*. We use *iperf* [62] to measure bandwidth.

* *getLossRate(IP1, IP2, [time, accuracy])*. Given two IP addresses *IP1* and *IP2*, return the packet loss rate from *IP1* to *IP2*. As loss rate is often small and hard to measure, *accuracy* specifies how accurate the returned result should be.

3. Control knowledge. The Internet consists of thousands of administration domains, and each domain defines its own policy on route announcements, firewall rules, etc. We provide the following examples on control knowledge:

* *isPortBlocked(port, AS)*. Given a port number and an AS number, this function returns whether that port is blocked by that AS.

* *getCost(IP1, IP2, metric)*. Return the cost of the path between IP1 and IP2 in term of the cost metric, such as monetary cost.

The function set above is just one example set. Additional network knowledge will be provided in future work, and more sophisticated functions can be built on top of the primitives. For example, given two pairs of IP addresses, we can use *getASPath* to determine if the paths between the two pairs of IP addresses intersect at some AS. This is helpful when

studying the interaction between traffic or building overlays with no overlapping paths. As another example, agents can collaborate with each other to monitor the networks, and locate possible failures in the network [130]. Therefore, we expect that more functions will be added to the NetKP as required by network management and applications.

3.3.2 Agent Organization in the NetKP

Organization Based on Network Topology

As the NetKP consists of agents widely distributed in the Internet, a natural way to organize agents is to follow the network topology. Here a region approximately corresponds to an administrative domain in the Internet, which is either a routing domain like an autonomous system, or an intranet such as the HP corporate network. Autonomous Systems are very heterogeneous, and in some cases we may need to divide agents in a large Autonomous System into several regions. We follow the following rules: First, agents are organized into regions based on network topology. Specifically, agents in the same Autonomous System are grouped into one region. Second, if a region covers too large an area or contains too many agents, it will be split into multiple smaller regions. Third, regions that contain only a few agents will be merged with a nearby region. Fourth, each region has a leader, selected from the agents under certain criteria, who is in charge of the communication with other regions. There are about 17,000 ASes in the Internet [99], so the number of regions in the NetKP will be in the same scale.

Within-Region Organization

Different regions may choose different organizations. We do not impose any specific organization within a region, because we believe that a proper organization should be based on many factors, including the number of agents in the region, how closely they collaborate with each other, the requirement on robustness, etc. An adaptive scheme is as follows. When the number of agents is small, they form a complete graph, but it is not scalable. As the number of agents increases, we can maintain the connectivity among agents by forming a connected graph. To do that, each agent connects to a number of nearby agents, and the

leader makes sure that the graph is not partitioned. Another candidate is a distributed hash table [104, 122, 109, 133].

Cross-Region Organization

Agents in different regions need to collaborate with each other in order to address the problems that span the regions. Unlike the organization within a region where each region may choose its own organizing structure, we need a unified approach among regions.

We have at least two options for the organization among the regional leaders. The first choice is a topology-based structure. In this structure, a region connects to other regions that are in the same AS and the neighboring ASes. Neighboring ASes include its providers, customers and peers. The second choice is a distributed hash table (DHT). DHTs provide a robust and efficient self-organizing infrastructure. We choose a structure that combines network topology and distributed hash tables. We will address this in detail in Chapter 4.

To resolve requests and disseminate knowledge, an aggregation and broadcast mechanism is needed at the region level. We will discuss this in detail in Chapter 5.

3.3.3 Agent and Region Operations

Agent operations and region maintenance are dependent on the organization within and among regions. As we do not impose any specific agent organization within a region, and will discuss cross-region organization in Chapter 4, we focus on the generic operations within a region and among regions in this section.

Joining

We assume there is a global agent that manages a directory service on the existing regions. This centralized global agent may be replaced by a cluster of agents. A new agent joins the system by contacting a well-known global directory agent and submitting its constraints. The directory agent matches the constraints with the regions in the region directory, and returns a number of regions that match the constraints. The new agent then compares its constraints with the collected regions, and decides which one(s) to connect to. Then the

new agent checks the latency between itself and the candidate regional leaders that are in the same AS, and chooses the closest one that matches its constraints. Then it registers at the regional leader, and retrieves from the leader a list of nearby agents to connect to. If no appropriate regions exist, the new agent will create a new region, connect to a number of neighboring regions, and register at the global directory. In this case, the agent automatically becomes the leader of the new region.

We realize that the global directory neither scales nor provides enough reliability. Therefore, instead of relying on the well-known global directory agent, an alternative is to design a distributed service. For example, we can set up authoritative agents in each network, just like the DNS system, and new agents can join the NetKP through a nearby authoritative agent. We will see that by using the mechanisms proposed in Chapter 4 and 5, the global directory is not needed.

Connection maintenance

A member agent periodically sends out heartbeat messages to its regional leader and nearby agents that it has connected to. If an agent does not receive acknowledgments from some of its neighbors, a repair procedure starts to find more neighbors to maintain the connectivity. If it cannot reach the leader, a new leader will be selected among the member agents according to certain criteria such as the capability and location. If one cannot reach the leader due to network partition, those agents who cannot reach the leader will form a new region. As discussed before, agent organization within a region can be done in different ways. In this work, each agent connects to a number of nodes, and the leader decides in a way so that all the agents in the region are connected, even if without the leader. This is important, as it allows all the remaining agents to find a new leader in case of leader failures. A repair may or may not be needed when one cannot connect to a neighbor, depending on whether this will cause a potential partition or not.

A regional leader contacts the directory service periodically, to let the service know the current status of the region. It also maintains connections with a number of other leaders, depending on the structure of the cross-region organization. A new leader is responsible for connecting to existing leaders. Leaders organize themselves in a way so that all the leaders

form a connected graph, as we will discuss in Chapter 4.

The directory agent is a rendezvous point. It passively maintains the region information. We do not place too many responsibilities on it to avoid the single point of failure.

Connection repair

Agent organization needs to be repaired when agents change their constraints or some neighbors leave. In the first case, an agent will ask the regional leader to find appropriate regions, or contact the global directory agent. In the second case, if an agent cannot contact some of its neighbors, it first checks if more neighbors are needed to maintain connectivity. If so, it will contact the current neighbors to find more agents, or ask the leaders.

If a regional leader becomes unresponsive, the agent who detects this notifies other agents in the region, and they work together to select a new leader within the region. As mentioned earlier, the new leader needs to reconstruct the region information, which may require broadcast to all the agent members.

The connectivity among leaders is maintained based on the way they are organized. If the number of leaders is small and they form a complete graph, then each change needs to be broadcast to all the leaders. If they form a distributed hash table, the connectivity is maintained according to the DHT protocol, as described in Chapter 4.

Note that we assume the global directory service is always working, so no maintenance or repair is needed for it. We can make it more robust by providing multiple directory agents.

Agent departure

There are two kinds of agent departure: graceful or not. In a graceful leaving, an agent notifies the regional leader, and the leader removes it from the list. If the departing agent is a regional leader, then other agents need to select a new leader, and the old leader hands over the region information to the new leader. If the departing agent is the last one in the region (and thus it must be the leader), then it will notify other leaders and the global directory agent.

For an ungraceful leave, if it is a member agent, this will be detected by its neighbors and the regional leader due to the heartbeat timeout. If it is a regional leader, then when a member agent notices it, a leader needs to be selected among agents in the region, and when another leader notices it, a repair is conducted. The new leader needs to reconstruct the region information including the agent membership in the region. If the departing leader is the only agent in the region, then this region will simply disappear, and be removed from the global directory.

Region Creation and Removal

There are two cases when a new region is created. As mentioned earlier, a region is created when a new agent cannot find an appropriate region to join, and the new agent becomes the leader automatically. A region may split into two smaller regions when the number of agents becomes too large. In this case, the current leader becomes the leader in one of the two regions, and a new leader will be selected for the other region. In addition to the above two cases, a region could be created because of network failures, which is discussed in region merging and splitting.

There are two cases when a region is removed. The first is when all the agents in it leave. Then the region is removed from the global directory agent. The second is when two regions merge. One of the two leaders will become the new leader, notify the neighbors, and update the region information at the global directory agent.

Region Merging and Splitting

As agents join and leave, regions may need to be split or merged to achieve a proper size. We use two simple thresholds: $T1$ and $T2$. If the number of agents in a region is fewer than $T1$, then the region can merge with a nearby region, as long as they are in neighboring Autonomous Systems. A new leader will be selected between two previous leaders in each region. If the number of agents is greater than $T2$, then the region should be split, based on network proximity. The previous leader continues to be the leader in a new region, and another leader will be selected in the region without the previous leader.

Once in a while, the network may be partitioned, and this causes complications in region

maintenance. Assume two partitions, A and B, are made within a region, and the regional leader is in partition A. Then all agents in A, including the regional leader, think they are fine although their neighbors in B seem to have left. Agents in B will think they need to select a new leader, under the same region ID. This will confuse the global directory agent as it sees two regions with the same region ID but different regional leaders. If the global directory agent just tolerates this, when the network recovers, there will be more confusions between agents in A and B. The global directory agent can ask the region with the new leader, B in this case, to change the region ID to form a new region.

In an alternative network partition case, two agents a and b may not be able to talk to each other, but another agent c may be able to talk to both of them. If two regions are going to be formed, then one solution for c is to choose a region to join.

In short, problems with individual agents are easier to detect and fix than network problems. We need a robust design mechanism to be able to either reconcile the inconsistency or tolerate it to some extent. Future research is needed in this issue.

3.3.4 Request Propagation in the NetKP

A key function of the NetKP is to resolve requests for network knowledge. We briefly discuss the way a request is resolved in the NetKP. Most requests for network knowledge can be resolved by following the network topology to find the agent who knows the answer. There have been other research efforts using similar approaches, such as iPlane [75]. In this thesis, we do not focus on this type of request, but on a broadcast and aggregation mechanism for requests that cannot be resolved by following the network topology and for initializing spec-KPs, as discussed in Chapter 5.

Requests for network knowledge can be issued by different agents. Agents in the NetKP issue requests to obtain certain knowledge so as to organize themselves into regions; agents in spec-KPs also issue requests for network knowledge to construct their efficient connectivity graphs, such as routing overlays, multicast trees, etc.

Requests can be about properties of different entities in the networks. A simple request example is the latency between two hosts, while complicated requests can be *isPortBlocked*

or the network condition, such as the degree of network congestion. In the latter case, sometimes it is not clear which agents are responsible for or can resolve the request, and an agent that receives the request may initialize the operation and work together with other agents to resolve the request. Such an operation can be complicated and costly, and may require authorization. Here I start with simple requests, and will address more complicated issues like broadcast in Chapter 5.

A request for simple network knowledge is resolved by the local agent as follows. For some requests, the agent resolves them directly and returns the responses to the requesters, without consulting other agents. This occurs when the local knowledge base already contains the necessary knowledge. Examples of such requests are *getASN* and *getASPath* when the source IP is in the local AS, as the local BGP table contains the mapping between IP addresses and AS numbers and the AS paths originating from the local AS to all the other ASes.

For requests that require non-local network knowledge, such as the AS path between two remote agents, the local agent forwards such requests to the agents that have the knowledge to get the answers. Because the local agent has the knowledge of AS paths from the local AS to all other ASes, it can figure out which agents in which AS(es) should be able to resolve the request. Therefore, it forwards the request following the corresponding AS path to an agent in a neighboring AS, and that agent again forwards the request to its neighboring AS following the same AS path until the request reaches an agent in the destination AS. The agent returns the answer to the source agent following the same AS path in reverse, and agents along the path may cache the answer. By following the reverse path instead of the default AS path and agents caching along the path, we construct an implicit aggregation tree, which helps to resolve similar requests more quickly in the future, because agents along the AS path may resolve the request if they happen to have cached previous answers. Figure 3-2 shows the resolution procedure.

So far we have briefly discussed how requests for network knowledge are resolved by following the network topology in the NetKP. Those requests all explicitly or implicitly specify the location where they should be sent. Not all the requests can be resolved in this way, such as a request to check if the local network is under intrusion attempts. Further-

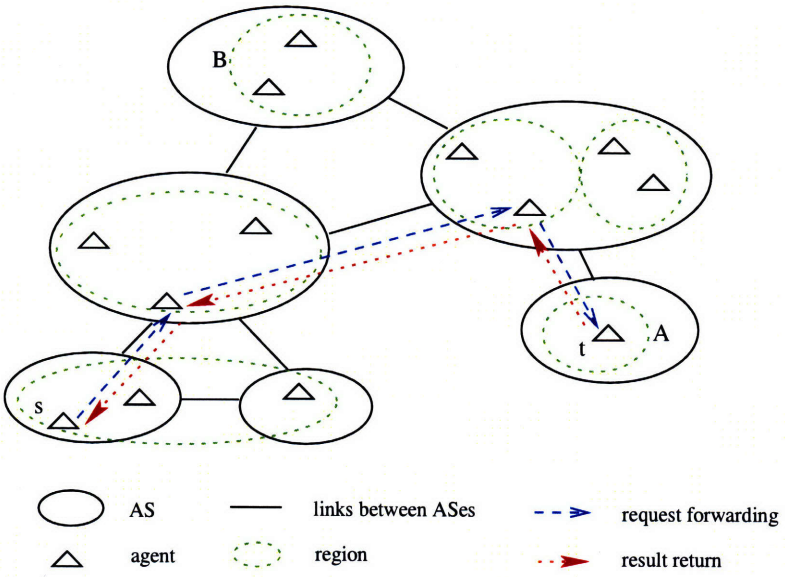


Figure 3-2: Request resolution in the NetKP. Agent s issues a request for the AS path between AS A and B. Agent s forwards the request to agents along the AS path, until it reaches agent t in AS A, as shown in the dashed arrow lines. Agent t checks its local knowledge base, and returns the corresponding AS path, as shown in the dotted arrow lines.

more, requests from spec-KPs about non-network knowledge are not resolvable by following the network topology, such as a request to find a specific intrusion detection technique. In Chapter 5, I focus on a global broadcast and aggregation mechanism among regions to propagate this kind of request.

3.3.5 Network Knowledge Collection

We briefly discuss how the three kinds of knowledge, configuration, operation, and control knowledge, are collected in our system in this section.

First, agents collect configuration knowledge from the local AS. To do so, agents need authorization to access BGP tables from local border routers. We believe that BGP information is usually not sensitive, so organizations may be willing to disclose this information, and agents can employ policies when exposing such information to others. Network topology is also stable, so it can be stored locally and updated infrequently. In this work, we use BGP data from multiple sources to construct an AS-level network topology as complete as possible, as was done by He et al. [56].

To provide configuration knowledge, especially network topology knowledge, some agents maintain two kinds of network topology knowledge at the Autonomous System level: (1) the AS number to which an IP address belongs; (2) the AS path between the local AS and another AS.

Some AS information is already available [17]. Other researchers have successfully retrieved BGP information. For example, PlanetLab [96] implemented an application support layer to provide BGP information. To do this, a PlanetLab server is matched with a BGP router. They are configured to provide a one-way information flow from the router to the PlanetLab node [81]. This does not require implementing any special interface to the routers. Furthermore, a set of servers in PlanetLab collectively construct a peering AS graph. A PlanetLab node also implements a PLUTO BGP Sensor interface to provide applications an easy access to BGP information [115]. Note that it is possible to replicate the service of the mapping between IP addresses and their AS numbers in each region, as the data set is not large. For example, the compressed data set of the RouteViews BGP tables is only 13MB.

In this way, an agent only maintains the local view of the Internet, which represents the reachability of the local network to the rest of the Internet. This is due to the following reasons: First, it is hard to get an accurate global view of network topology, but it is easy to obtain the local view; Second, such local network knowledge should be able to satisfy local requests most of the time; Third, remote network knowledge can be obtained from other agents through request resolution. Note that restrictions may be applied to the parameters of those functions. For example, an application may not be allowed to query the AS path between any two ASes, due to the privacy concerns of routing information of those ASes.

Geographic information is another kind of network knowledge we are interested in. Geographic information provides physical location information, which is often directly related to network performance, such as latency. It also enables a large set of location-aware applications. It is not trivial to obtain accurate geography information today, but approximate location is enough to organize agents in this work. We use data from the GeoIP Database [41] for this purpose, and plan to leverage existing techniques like [90] in the future. Agents can run the geography service, and register the service at the regional

leaders. However, it gets complicated if an agent is mobile. We set aside the mobility issue in this work.

Second, operation knowledge may be more costly to obtain and maintain than configuration knowledge. Among different kinds of operation knowledge, latency is usually easy and lightweight to measure using measurement tools such as *ping*. Unless the latency to a large number of hosts or an average over a long time period is needed, real-time measurement will work because of its simplicity and low overhead. Other information, such as bandwidth and loss rate, can be obtained through measurement with more overhead. Many tools have been developed to measure network status, and new tools are being developed. Agents can use those tools and share performance knowledge. A request for performance knowledge between two hosts is resolved by agents near the hosts. For example, the latency between two hosts can be approximated by the latency between two agents plus the latency between each host and its nearby agent, similar to [55]. As another example, agents may infer the property of a new path by segmentation and composition using previous measurement results. This is similar to network tomography [28]. Consistency is another important issue here. Agents at different locations may return different answers to the same request, and the same request asked at different times may get different answers, even if it is not time sensitive.

Note that the performance and geographic knowledge may be approximate instead of accurate. First, the performance knowledge changes frequently. Even if we obtain accurate measurement results, it may be outdated when it is returned to the requester. Second, in many situations it is enough to have approximate information. For example, a streaming video application only needs to know the class of bandwidth (high, medium, low) to determine the appropriate encoding method.

Third, control knowledge, such as policies, is usually very hard to obtain directly or infer. Sometimes we may discover it partially. For example, we can test if an ISP blocks any port. But generally this knowledge is hard to observe from outside. We focus on simple policy information in this thesis.

3.4 Spec-KPs for Network Management and Applications

For the purpose of network management and applications, we propose to build multiple spec-KPs, one for each specific service. In this section, I discuss briefly the general issues in the spec-KP organization. Note that in the following discussion, without special specification, agents refer to those in the spec-KPs, not in the NetKP. I will present two case studies on the spec-KPs in detail in Chapter 6 and Chapter 7.

3.4.1 Constraints on the Spec-KP organization

Similar to the NetKP, agents in the spec-KPs are organized into regions, but under a different set of criteria. Agents in the spec-KPs register their information in the NetKP, but may not participate in the activities in the NetKP.

The set of orthogonal constraints for organizing the spec-KPs fall into five categories. The primary one is the functionality that defines the spec-KP and how the functionality is partitioned. A spec-KP is likely to, although not necessarily, run on multiple agents to achieve the functionality by composition. Those agents either do the same task at different places or participate in different parts of the problem. For example, if we design a diagnosis system for the DNS system, the naming hierarchy and the zone structure will inevitably play an important role in such a spec-KP. As another example, intrusion detection itself does not impose any constraints on the corresponding spec-KP, but due to security and privacy reasons, such a system is often organized based on corporate network boundaries.

A second and obvious constraint is the physical location and network topology. When a spec-KP is to be instantiated, the authority on whose behalf it is happening may have an interest in constraining it to run only in some part of the network or in some other location based region, such as geographical. For example an ISP network manager might want to run a particular spec-KP within the scope of the ISP network, simply due to network topology reasons. A simple version of this may be AS or IP based. In terms of geography, such a specification may be as general as named geographic region or as specific as ranges of longitude and latitude.

The third kind is external policy constraints. They are security policies, pricing or

economic policies, and other incentive-based policies. Security policies specify hard constraints on which information and functionality can and cannot be made available across specified boundaries and by whom. Pricing constraints allow for perhaps a sliding or degree based decision. Other forms of incentives may be designed specifically to encourage cooperation, in the face of proprietary and other security constraints.

The fourth kind is performance. Applications may have to run with a set of efficiency criteria, which may determine the placements of agents. For example, if it is important to have low latency, then paths between agents should be selected on that basis; if the amount of network traffic should be low, then agents may be collocated on the same nodes as much as possible.

The fifth kind is shared resource usage. This requires that the spec-KPs, in determining their organizations, know enough about others which may be sharing resources to make a possible compromise in order that the spec-KPs not interfere unnecessarily with each, as best possible. We take a lead from previous work, beginning with a set of information similar to that of CoMon [93] from PlanetLab. CoMon provides a monitoring statistics for PlanetLab at both a node level and a slice level. It can be used to see what is affecting the performance of nodes, and to examine the resource profiles of individual experiments. In the longer run, it will be necessary for this sort of information to be distributed in the KP, unlike what is currently being built in PlanetLab.

3.4.2 Agent Organization in Spec-KPs

Generic agent and region operations in the spec-KPs are similar to that in the NetKP. Within a region, agents can be organized in different ways, depending on the size of the region and the distribution of the distance metrics. When the size is small, a full mesh will suffice. That is, each one knows all the others. When the size is large, a structured peer-to-peer organization, such as distributed hash table, may be preferred. In general, we want to limit the size of a region so as to simplify its organization and recommend a full mesh structure. There is no single answer to the region size. It will depend on what they are doing under what conditions. Cross-region organization in the spec-KPs depends on the nature of the

spec-KP itself. In Chapter 6 and 7, we present two case studies to demonstrate different organizations for different tasks.

An important issue in the spec-KPs is how to discover agents in the same problem domain when an agent joins. There are two steps. The first is to find a spec-KP that matches its interest, and if no such spec-KP exists, to create a new spec-KP. This can be done using the broadcast mechanism presented in Chapter 5 to broadcast the request to all the regional leaders in the NetKP. Note that all the spec-KPs register their information in the NetKP.

The second is to find an appropriate region after the spec-KP is found or formed. If there are multiple regions formed in order to partition the set of agents, when an agent comes into existence, the choice of how regions are defined is going to drive which region the agent will end up with. In this work, the discovery procedure uses the underlying NetKP. Each region in the NetKP maintains a list of local agents in the spec-KPs and their interests. A request looking for agents with similar interests is propagated from the local region to neighboring regions. In most cases there is already a spec-KP with this interest, and the new agent only needs to find one that is already in this network to join the spec-KP. Furthermore, a high-level task may involve multiple agents belonging to different spec-KPs. For example, to diagnosis a web access failure, we need help from agents on DNS diagnosis, routing diagnosis, server status monitoring, etc. To find those agents, we need to first find local agents in the NetKP, and then search for agents in each spec-KP.

3.4.3 Request Propagation in Spec-KPs

Request resolution in a spec-KP depends on the nature of the spec-KP itself. A simple example is as follows. When an agent issues a request, it is propagated to all agents in the same region, and also to one or a few agents in every region within a certain number of *region hops*, say n. When a piece of knowledge is issued, it is also disseminated in the same way. Therefore, the knowledge within n region hops is guaranteed to be found by a request. Figure 3-3 shows an example with n=2. This shows how regions help to improve scalability and efficiency. To further extend this, we can consider many factors, such as the location

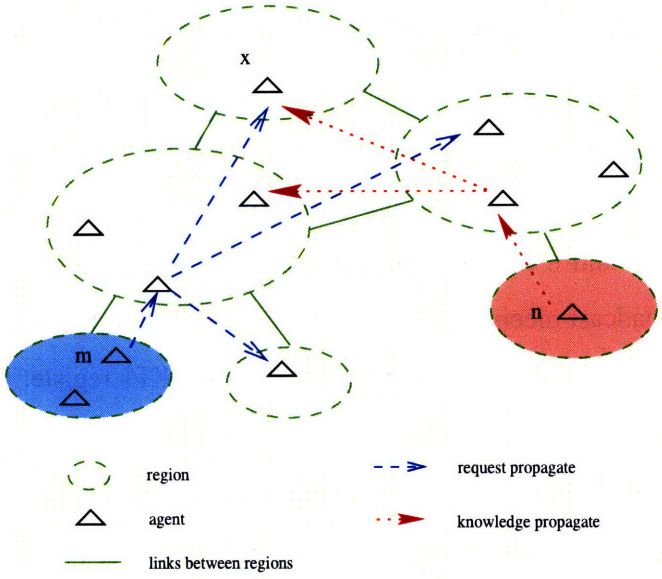


Figure 3-3: Request and knowledge propagation in the spec-KPs. This shows a request from agent *m* and a piece of knowledge from agent *n* meet at agent *x*.

of an agent, the cost model, and even the diameter of a specialized KP, etc. In Chapter 6 and Chapter 7, I present different propagation mechanisms based on the functionality and constraints of the spec-KPs.

3.5 Summary

In this chapter, I discuss the design criteria and principles for the knowledge plane, and propose an architecture that consists of the NetKP and multiple spec-KPs. Based on the criteria and principles, I design a region-based agent organization, and discuss the operations of agents and regions. The insights of the NetKP are twofold. One is that valuable knowledge at the network level is exposed to network management and applications. The other is that the overhead of this exposure is amortized among network applications. To do so, we need an efficient agent organization. In the next two chapters, I address two key issues in the NetKP: cross-region organization, and broadcast and aggregation among regions, both in the NetKP.

Chapter 4

Cross-Region Organization

In this chapter, I focus on a network-topology-based distributed hash table for cross-region organization among the regional leaders, and present a hybrid neighbor selection mechanism using Autonomous System information. We find that, by combining the distributed hash table and network topology knowledge effectively, we can have a scalable, efficient, robust and non-intrusive organization among regions.

4.1 Design Rationale

Cross-region organization is a core issue in the NetKP, as agents in different regions need to collaborate with one another in order to address the problems that span the regions. Unlike the organization within a region where each region may choose its own organizing structure, we need a unified approach among regions.

As mentioned in Section 3.2, four properties are important to the agent organization in the NetKP: efficiency, scalability, robustness, and non-intrusiveness. Section 3.3.2 briefly discusses two options for the organization among the regional leaders, and we elaborate them here. The first is a network-topology-based structure. In this structure, regions connect to each other following the network topology. Specifically, a region connects to other regions that are in the same AS and the neighboring ASes. The neighboring ASes include its providers, customers and peers, or ASes that are closest in any of the three kinds. The advantage of this approach is that this structure follows the network topology naturally, and

thus is efficient for aggregating information and suppressing redundant requests. However, there are several problems with this structure. First, this approach may lead to an unbalanced structure. We know that the Internet topology at the Autonomous System level can be described efficiently with power laws, where some ASes have many neighbors while many other ASes have only one or a few neighbors [116, 123]. As a result, this approach may lead to a structure in which some regions may have to connect to a large number of regions, while some can only find a few. For example, a region in a top-tier AS may have hundreds of neighboring regions, while a region in a small bottom-tier AS may only connect to its provider. Second, it is hard to maintain this structure in case of churn. If a region disappears due to agent departures, region merging or network partitions, its neighboring regions will need to discover other regions to maintain the connectivity. Third, when the regions are sparse in the Internet, the connectivity of this structure is not well defined. For example, if there are no regions in the provider's AS, a region needs to find other regions in two or more AS hops. All these problems can be fixed, but obviously a more robust and clean design is needed.

The second choice is a distributed hash table (DHT). Distributed Hash Tables, such as CAN [104], Chord [122] , Pastry [109], and Tapestry [133], provide a scalable and robust organization, in which any information is guaranteed to be located in a limited number of steps (usually $O(\log n)$). These systems provide a robust self-organizing infrastructure that is expected to become the fundamental component of large-scale distributed systems. However, a pure DHT structure is not enough for our purpose.

DHTs provide scalability and robustness, but they often rely on active probing to achieve efficiency. Most topology-aware DHT lookup algorithms proposed so far, such as proximity neighbor selection and proximity routing [54], require each peer to probe other peers directly to discover proximity. Such probing generates a considerable amount of network traffic. Similarly, many network applications and services, such as end-system multicast [58], DHT-based DNS [103], and content delivery networks [12], require efficient organization among the participants, and several previous research efforts focus on constructing network overlays in order to route traffic optimally, such as RON [16]. They all rely on active probing using ping or traceroute to measure path quality and to detect anomalies

[130, 39]. As a result, 1GB of ping traffic was observed daily on PlanetLab in 2003, which equals to one ping per second per node [82]. As another example, RON periodically measures all inter-node paths to discover link failures; due to its probing traffic, a 50-node RON requires 33Kbps bandwidth for each node, which prevents it from scaling to a large size. As overlay networks grow in popularity and sizes, this may lead to a significant increase in network traffic and contention for network resources, which may cause network instability.

We choose a hybrid structure that combines network topology and distributed hash tables. DHTs provide scalability and robustness, and we need a design for efficiency and non-intrusiveness. In this chapter, we demonstrate the use of network topology knowledge to improve the efficiency of the DHT-based cross-region organization, while maintaining low overhead. Specifically, we design a hybrid proximity neighbor selection algorithm that uses Autonomous System information to estimate network latency. Proximity neighbor selection (PNS) in DHTs is, given a number of neighboring leaders, which ones a node should choose as the neighbors in its DHT routing table. This is an important issue, as it determines the efficiency of the region-level organization among leaders. We use the AS-path length as a proxy for network latency to filter out unlikely candidates without probing them during the selection process, and only a small number of leaders who pass the filtering will be probed. Compared with those approaches based on active probing, our algorithm can significantly reduce the amount of probing traffic without greatly undermining DHT lookup performance, and our savings on probing traffic increase with network size, as demonstrated in the experiments. Note that in this organization, each regional leader also maintains a number of its topological neighboring regional leaders, as it is convenient to resolve the requests that follow network topology whenever possible, but this list of neighbors is not required.

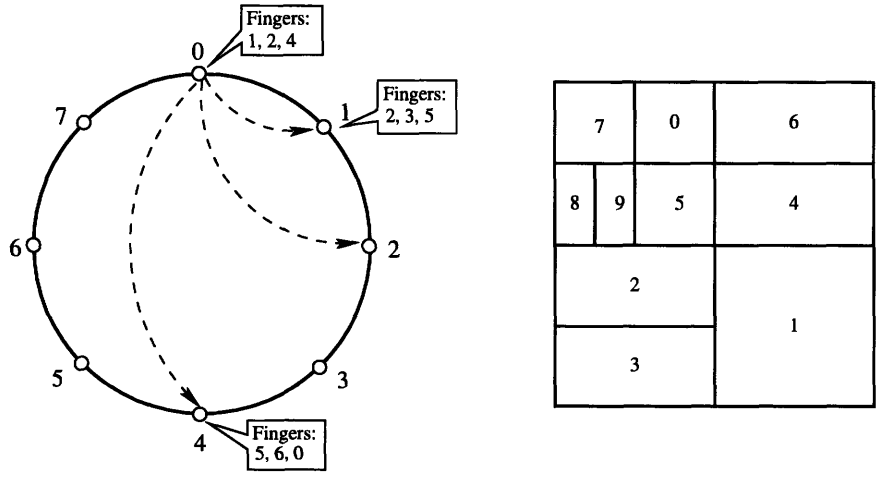
The rest of the chapter is organized as follows. Section 4.2 and 4.3 present an overview on distributed hash tables and the Internet topology, respectively. Section 4.4 presents a hybrid PNS algorithm using network topology information. Section 4.5 evaluates the performance of our approach. Section 4.6 discusses the policy implication of our approach. Section 4.7 reviews related work. We conclude in Section 4.8.

4.2 Distributed Hash Tables

4.2.1 DHT Routing

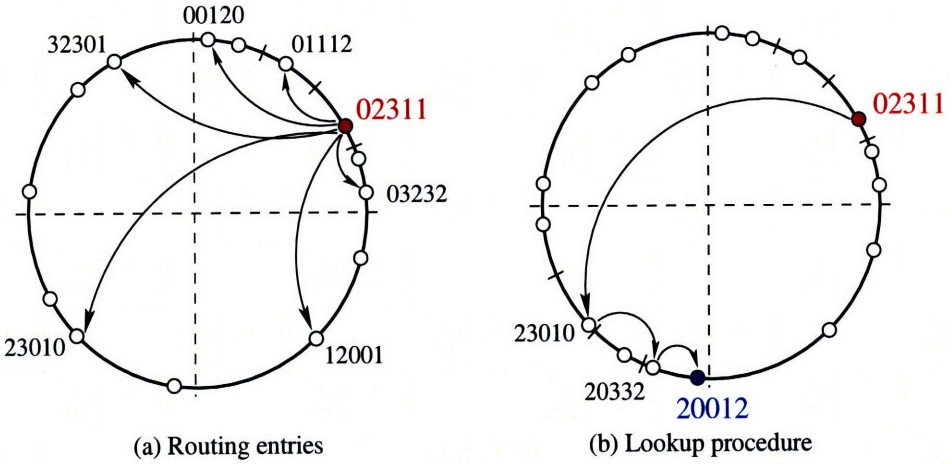
The explosion of peer-to-peer applications, originally file sharing such as Napster [85], Gnutella [49] and BitTorrent [23], has inspired the more structured peer-to-peer networks. Structured peer-to-peer networks, such as CAN [104], Chord [122], Pastry [109], and Tapestry [133], provide a scalable distributed hash table, in which any piece of data is guaranteed to be located in a limited number of steps. These systems provide a robust self-organizing infrastructure that is expected to eventually become the fundamental component of large-scale distributed applications. In this section, we present some background on structured P2P networks and the standard proximity neighbor selection.

DHTs provide an efficient and scalable routing mechanism where a lookup operation can be performed in typically $O(\log n)$ steps and each node maintains only typically $O(\log n)$ states, where n is the number of nodes. Many geometries have been proposed in DHTs to achieve the balance between the number of states and the number of lookup steps [54]. For instance, CAN uses a hypercube [104], Chord uses a ring [122], Viceroy forms a butterfly network [77], and Pastry combines a tree and a ring [109]. Figure 4-1 demonstrates two examples.



(a) Chord (b) CAN

Figure 4-1: DHT geometry examples. (a) shows the finger tables of Chord, and (b) shows how nodes divide the name space in CAN. The numbers are the nodes' IDs.



(a) Routing entries (b) Lookup procedure

Figure 4-2: Pastry routing. In (a), the circle represents the name space, and the arrows show the entries in the routing table of node 02311, which is also shown in Table 4.1. (b) shows a lookup procedure by 02311 to find node 20012.

We use Pastry [109] as an example to demonstrate the DHT routing. Pastry is a generic P2P infrastructure. Each Pastry node has a unique ID in a circular 128-bit name space. A node is responsible for storing data whose keys are between the present node and its predecessor. All nodes together form an overlay network, and a lookup query is routed through the overlay to a destination by a series of forwarding steps based on routing tables. Each node keeps track of its neighboring nodes that are closest in the name space in its leaf set.

For the purpose of routing, each node maintains a routing table that contains a number of node ids and their IP addresses. Node IDs are considered as a sequence of digits in base 2^b. A routing table is organized into $\frac{128}{2^b}$ rows and 2^b columns. The entries in row i refer to nodes who share the first i digits with the present node, and the $(i+1)^{th}$ digit of a node in row i and column j equals j. Table 4.1 and Table 4.2 shows simplified routing table examples of two nodes. Figure 4-2 demonstrates the routing geometry of Pastry and a lookup procedure. In Figure 4-2b, 02311 checks its routing table to find 23010, which has the longest matching prefix with 20012, and forwards a lookup message to it. 23010 checks its routing table for a longest prefix match to 20012, which is 20332. It is guaranteed to find a node with longer matching prefix, due to the routing geometry of Pastry. This process continues until 20012 is reached.

X	12001	23010	32301
00120	01112	X	03232
02011		02231	X
	X		02330
	X	02312	

Table 4.1: A Pastry routing table of node 02311. In this example, the name space is 10-bit, and each digit consists of 2 bits. X refers to the entries that match the present node's ID. Some entries are empty because the present node has not found appropriate nodes for them. The IP address of each node is not shown here.

02311	12001	X	32301
20332	21003	22102	X
X		23211	
23001	X		23030
X		23012	

Table 4.2: The routing table of node 23010.

X	1****	2****	3****
00***	01***	X	03***
020**	021**	022**	X
0230*	X	0232*	0233*
02310	X	02312	02333

Table 4.3: A generalized Pastry routing table of node 02311. In this example, the name space is 10-bit, and each digit consists of 2 bits. $03***$ means that a node whose ID begins with 03 fits that entry. X refers to the entries that match the present node's ID.

4.2.2 Proximity Neighbor Selection

Although given a DHT geometry the average number of steps is fixed, the actual latency of a lookup is also determined by the latency of each step. Proximity neighbor selection (PNS) is an effective method to reduce the lookup latency. PNS is based on the fact that multiple peers may be suitable for a routing table entry. Table 4.3 shows the flexibility of the routing table entries. Note that the flexibility is the highest at the top row, and decreases to none at the bottom row. In PNS, peers adjust their routing table entries for proximity by periodically exchanging them with its neighbors (those in its routing table). After getting multiple candidates, the present node probes all the candidates to find the closest ones to fill in its routing table entries. PNS requires the node to actively probe all candidates, which generates significant network traffic. We use Pastry as an example. Suppose the network size is N, and each entry receives k candidates on average. Since the routing table size is $O(\log N)$, the total number of probing messages of the network is $O(k\dot{N}\dot{\log}N)$ during each neighbor selection period. This situation would only deteriorate with the network sizes.

4.3 Network Topology

As discussed in Chapter 3, we classify network knowledge into three kinds: configuration knowledge, operation performance, and control knowledge. Configuration knowledge, such as network topology, is relatively static and coarse-grained. Operation knowledge, such as network performance, is fine-grained and dynamic, and represents accurate condition of the Internet, and may change dramatically over time. Control knowledge refers to the policies imposed by the network providers.

This classification helps us think about how cross-region organization can obtain and take advantage of different types of knowledge in different ways. Operation knowledge provides accurate network conditions, but usually requires active probing, and thus adds to the Internet traffic. On the other hand, configuration knowledge can be provided without real-time measurements, as it is more stable. In this section, we discuss the accuracy of the AS-path length as a proxy for network latency.

4.3.1 AS Topology Information

Interconnected ASes together comprise an AS-level Internet topology. However, the AS-level topology cannot be simply drawn as an undirected graph, because there are several complications. First, there is a large number of ASes; it is difficult to collect complete, accurate, and timely topology information. Second, AS links are not undirected. For example, a multi-homed customer AS will not transit traffic between its providers. Third, AS connectivity is not transitive. For example, suppose there are three ASes, A, B and C. A and B peer with each other, and B is C's customer. In this configuration, usually B will not accept traffic from A whose destination is C. Fourth, network errors such as link failures may affect the actual AS topology, and it is hard to detect them in a timely fashion.

For the first problem, it turns out that BGP routing tables at multiple vantage points already contain most ASes of the Internet [82, 123]. For example, RouteViews data contain more than 15,000 ASes [99]. To deal with the second problem, instead of using an undirected graph, we can construct the AS topology as a directed graph. In this way, we can distinguish the in-degree and out-degree of an AS, which is useful when considering replica placement, as discussed in next section. Furthermore, the second and the third problems interact with each other. For example, in the same AS scenario as that in last paragraph, B's out-degree is 2, if A and C are B's only neighboring ASes. However, A usually only accepts traffic from B whose destinations are in A or A's customers, because the link between A and B is a peering link; therefore, the out-degree 2 is deceptive in that B's links to its peer A and its provider C are not equal in their importance of providing connectivity. Fortunately, peering links are much fewer than provider-customer links. According to [123], only 4.75% links in the Internet are peering links, and many of them are links between backbone ASes who have no providers. Therefore, we believe this problem is minor. In the current work, we do not treat peering links specially in the AS topology, and we plan to consider this problem in the future work. Finally, it is not necessary to construct a completely accurate AS-level topology before we are able to use this information.

There are several advantages of using AS information. First, under our assumption, AS information can be obtained locally, which adds little to Internet traffic. Second, such

information is usually stable and static, and thus can be statically stored, easily retrieved, and infrequently updated. Third, AS information can be collected in an aggregate way by the local server and provided to clients, so that applications do not have to probe the Internet individually.

4.3.2 Relationship between AS-path and Latency

BGP [107] was designed for reachability and policy rather than routing efficiency, but previous research shows that the AS-path length correlates with network latency to some extent [88, 4]. An intuitive reason is that the longer an AS path is, the more ASes a packet has to go through. This is also because routing within an AS is usually efficient and well managed, and many anomalies occur on the links between ASes [6]. We reconfirm these results using the RouteViews data [99] and the Skitter data [3]. We retrieve end-to-end latency information from the Skitter data. For each pair of source and destination in the latency information, we search the RouteViews data to find the corresponding AS path using IP prefix matching. Figure 4-3 shows the relationship between the number of AS hops and latency. We also sum up the percentage of different AS-path lengths based on the Skitter data. We can see that the average latency is positively related to the lengths of AS paths. Note that the maximum number of AS hops in the Skitter dataset is 7, but the maximum number of AS hops in the Internet will be larger. In this figure, the average latency of 7 AS hops is lower than that of 6 AS hops, because the amount of data for 7 AS hops is very small and the average latency for it may not be very representative.

However, what is not shown in Figure 4-3 is the standard deviation of latencies with the same AS-path length, which turns out to be very large. The correlation coefficient between the raw AS hops and the latency is 0.24. That means we can hardly use the AS-path length directly as a proxy for network latency. There are several reasons why the AS-path length does not reflect latency very well. First and most importantly, ASes are very heterogeneous in terms of geographic size. A large AS may span one continent, so one such AS hop contains many router-level hops, and its latency may be large. In contrast, the latency of a small AS hop is much smaller. Even worse, if two nodes reside in two small neighboring

ASes, the latency between them may be lower than two distant nodes in a large AS. In that case, 1 AS hop is shorter than 0. Second, AS-path lengths are very coarse-grained. The AS-path lengths vary from 0 to tens of hops, while network latency varies from a few milliseconds to a few seconds. These two are the most important factors that affect the accuracy.

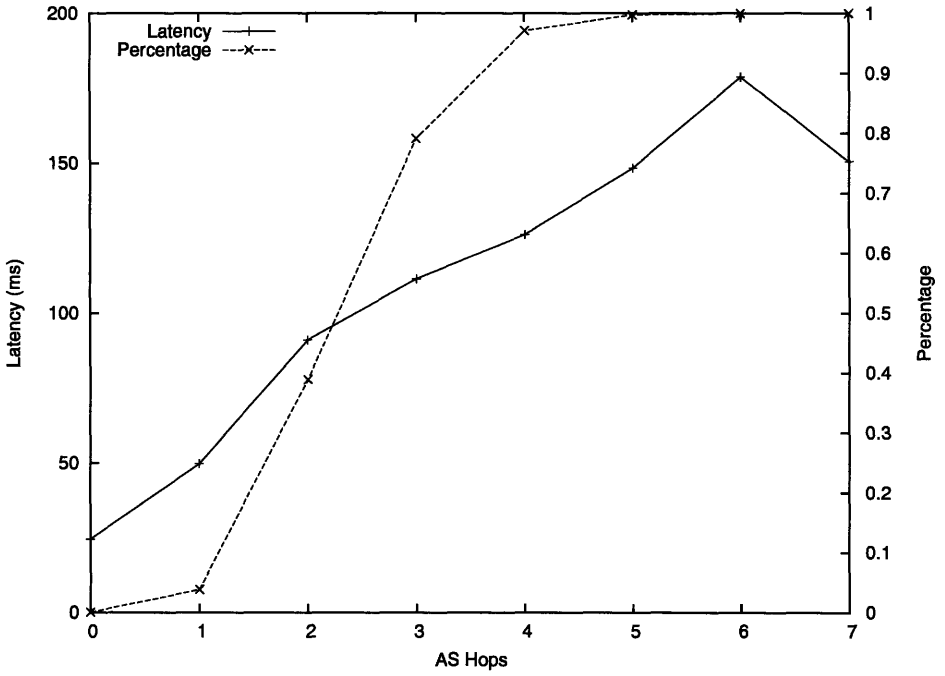


Figure 4-3: Relationship between AS hops and latency.

4.4 Leveraging Network Knowledge

In order to validate our proposition that there is benefit to be gained for the cross-region organization by using network topology information, we explore how to improve the lookup performance without incurring significant amount of probing traffic. Our goal is to maintain lookup performance comparable to the standard proximity neighbor selection algorithm, but to greatly reduce the network traffic generated by probing.

4.4.1 Hybrid Proximity Neighbor Selection

Algorithm

We propose a hybrid proximity neighbor selection algorithm in this section. The key idea is to use the AS-path length as a proxy for the end-to-end latency, so that a node does not have to probe all the candidates during proximity neighbor selection. As in the standard PNS, nodes exchange routing table entries periodically. After receiving multiple candidates for each entry, a node performs the hybrid PNS algorithm.

Our hybrid scheme consists of two steps. First, when a node receives several candidates to fill in a routing table entry, it does not probe all of them directly. Instead, it first calculates the AS-path length between itself and each candidate. Then it uses the AS-path lengths to filter out those that are unlikely to be close, leaving a much reduced set of candidates. Filtering candidates can be done in many ways, as described soon.

Second, the node probes the remaining candidates to find the nearest one. The closest node found is used for the corresponding routing table entry. This is the same as the standard PNS. Algorithm 1 shows the pseudo code of our hybrid PNS algorithm.

Filtering Functions

As mentioned in Section II, there are several complications that make the AS-path length not a completely accurate proxy for latency. To address that we design filtering functions. Filtering functions are used to deal with the problems that the AS path cannot address, such as network dynamics. In Algorithm 1 , the filtering process is implemented as a separate procedure in which different filtering choices are made. There are many options for the filtering functions, each of which has its own filtering criteria. Two extremes of the filtering functions are presented as follows. One is to select nodes solely on basis of the AS-path lengths. In this case, we can choose those candidates with the shortest AS-path length. If there are multiple nodes with the shortest AS-path length, we pick up one randomly from them. That is based on an optimistic belief that the shortest AS-path length corresponds to the shortest latency. In that case, we do not have to probe any candidate, and thus save all the probing traffic. However, the found candidate may not be the closest

Algorithm 1 The hybrid proximity neighbor selection algorithm.

```
owner: the present node that performs the selection.
N: a set of candidates to be selected.
A: an array of AS-path lengths between the present node
and the candidates.
R: the set of candidates after the filtering.
getASPathLength(i,j): return the AS-path length between
nodes i and j.


Node hybridPNS (){
  // first step
  For each node n in N,
    A[i] = getASPathLength (owner, n);
  NodeSet R = Filter (N, A);
  // second step
  min = infinity;
  For each node rR {
    latency = ping(r);
    if (latency < min) {
      min = latency;
      selected = r;
    }
  }
  return selected;
}

NodeSet Filter (NodeSet N, int[] A){
  threshold = the shortest AS-path length among all the candidates;
  For each n in N,
    if (A[i] == threshold)
      R = R+{n};
  return R;
}
```

one. The other extreme is to ignore the AS-path length information and to probe every candidate, i.e., the standard PNS algorithm. It can always find the closest candidate, but requires the maximum number of probing messages.

Between the two above extremes, there are many other choices. For example, in Algorithm 1, the filtering function is defined to choose those candidates with the shortest AS-path length and filter out the others. More sophisticated schemes can be employed. In our simulation, one of the filtering functions tested is defined to be that all candidates with the shortest AS-path length or an AS-path length that is only one more than the shortest length will pass the filtering. The assumption in this filtering function is that due to the heterogeneity of ASes, candidates with one more hop than the shortest one could still be the closest, so we do not filter out such candidates. But if a node's AS-path length is two or more than the shortest one, it is very unlikely to have a shorter latency.

There is a tradeoff between the lookup performance and the amount of probing traffic. The more the nodes are filtered out, the less traffic the probing will generate, but the more likely that the finally found node is not the closest one. However, the found node will usually still be very close to the present node. That provides the flexibility for each node to decide the efforts it wants to take to improve the lookup performance, considering its bandwidth restriction.

4.5 Performance Evaluation

In this section, we present the simulation results of our approach by comparing them with those currently used in the DHTs. We first describe the simulation setup. Then we present the lookup performance in terms of the latency, the number of AS hops, and the amount of the traffic, respectively.

4.5.1 Experiment Setup

We employ the Pastry structure among the leaders. Our simulation is configured in the same way as Pastry, with keep-alive messages for leaf sets and lazy routing table repairs [5]. The simulation is divided into discrete time periods. In each period N queries are

issued, where N is the network size. The querying node and the lookup ID are randomly chosen in the name space. Every 10 periods leaders exchange their routing table entries to perform proximity neighbor selection. To do this, each node sends each row in the routing table to neighbors in that row. The leaf set is set to 16, and each node probes all its neighbors in every period.

We evaluated our schemes on both synthetic Internet topologies and a topology generated from the real Internet data. The synthetic topologies are generated using the BRITE Topology Generator [48]. The topologies are two-layer networks. The first level is the AS level, and the second is the end host level. The BRITE Generator assigns latency to each link. Due to space limitation, only the results on one synthetic topology and the real topology are shown in this chapter. That topology consists of 150 ASes interconnected using the Waxman model and each AS contains 100 nodes interconnected in the power law model. The topology is undirected, and the routing between ASes is the shortest AS-path routing.

Second, we generated a real Internet topology from the RouteViews data and the Skitter data. We calculated the weight vector using the two data sets, and normalize the weight to be between 0 and 1. We built an Internet topology containing about 10,000 hosts by randomly choosing nodes in the Skitter data. We also generated an undirected AS-level topology and a database that contains the mapping between IP prefixes to AS numbers. For the host pairs appeared in the Skitter data, we infer the AS path between two hosts using the traceroute information; for other pairs, we use the shortest AS path as an approximation. In this way, we can find the AS-path length between any pair of nodes. Note that we make two simplifications here. In reality, AS links are directed, and the policy-based AS-level routing does not always match the shortest AS-paths. Since it is not our focus here to construct directed AS topologies with commercial relationships and to generate policy-based routing, we make the simplifications and plan to construct more realistic topologies in our future work.

Name	Best	Looser	NoProbe
Filter	If the AS-path length between a candidate and the present node is less than 2, such a candidate will pass the filtering, and others are filtered out; otherwise, if all the AS-path lengths are more than 1, then only those with the shortest AS-path length will be probed.	All candidates with the shortest AS-path length or an AS-path length that is only 1 more than the shortest length can pass the filtering; others are thrown away.	Only the AS-path length is used to distinguish candidates. If multiple nodes are with the shortest AS-path length, then a random one is chosen.
Naming	In the evaluation we found that it achieved a good balance between the lookup performance and the probing traffic. So it is called $Best$.	It allows more candidates to pass the filtering than $Best$, so it is called $Looser$.	This scheme does not require any probing, so it is called $NoProbe$.

Table 4.4: Filtering functions.

4.5.2 Organization Efficiency Evaluation

As lookup is the key function in DHTs, we compare different lookup schemes in this section. The goal is to show that our hybrid PNS schemes can achieve lookup performance comparable to the standard PNS, but with much less probing traffic.

Five schemes are compared: Raw, PNS, and three hybrid PNS schemes with different filtering functions. The Raw scheme implements a generic Pastry network without any optimization, in which a node sets up its routing table with randomly chosen nodes. This is the basis of comparison. PNS refers to the standard proximity neighbor selection used in Pastry. $Best$, $Looser$, and $NoProbe$ refer to three hybrid PNS schemes that use the AS hops but with different filtering functions, as described in Table 4.4. They are different in the strictness of filtering. Besides, SP (Shortest Path) is the direct end-to-end metric between the source and the destination.

Table 4.5 summarizes the performance of PNS and the three hybrid PNS schemes, on the synthetic topology and the real topology with both 10,000 nodes. We can see that

Network Size	Filter	Latency (ms)	AS Hops	Probing Traffic
Synthetic 10,000 nodes	PNS	78.4	7.03	100%
	Looser	78.8	6.94	22.2%
	Best	80.2	6.53	8.0%
	NoProbe	90.1	6.46	0%
Real 10,000 nodes	PNS	143.5	11.8	100%
	Looser	159.6	7.52	67.2%
	Best	173.9	5.72	11.9%
	NoProbe	296.3	4.77	0%

Table 4.5: Summary of lookup performance.

on both topologies, the order of lookup latency is, from low to high, $PNS < Looser < Best < NoProbe$. The order of the number of the AS hops per lookup is the same as the order of the traffic percentage, which is, from low to high, $NoProbe < Best < Looser < PNS$. On the synthetic topology, $Best$ and $Looser$ both perform very well in terms of lookup latency and save significant probing traffic.

Figure 4-4 to 4-9 show in detail the average lookup performance of different schemes under the network sizes from 1,000 to 10,000 nodes on the two topologies. Note that the performance does not significantly downgrade with the network size, as nodes are randomly chosen from the same node set.

Figure 4-4 and 4-7 show the average lookup latency on the two topologies. SP (Shortest Path) refers to the average end-to-end latency between the source and the destination. In both topologies, the Raw scheme performs the worst; its latency is much higher than all the others. PNS performs the best (SP is not regarded as a scheme). Between Raw and the PNS, the lookup latency, from low to high, is $Looser$, $Best$, and $NoProbe$. $Best$ and $Looser$ perform very close to PNS, especially in Figure 4-4, where they are hardly distinguishable. In Figure 4-7, the average lookup latency of $Best$ is about 20% higher than that in PNS. $NoProbe$ performs worse than the other two filtering functions, since it only uses AS-path lengths to choose nodes. From the results, we can see that the hybrid PNS schemes with a proper filtering function can significantly improve the lookup performance compared with Raw, and some are almost as effective as the PNS scheme.

Figures 4-5 and 4-8 compare the average number of AS hops in a lookup. SP refers to the length of the direct AS path between the source and the destination. The hybrid PNS schemes achieve shorter AS-path lengths than PNS does, since they take the AS-path length into consideration when filling the routing table. The average numbers of AS hops of all schemes except Raw are similar to each other on the synthetic topology, but they are quite different on the real topology. On the real topology, PNS requires about 9 AS hops per lookup when the network size is above 8000. In contrast, $Best$ requires about only 5.5 AS hops. A benefit of shorter AS paths is to reduce the possibility of lookup failures due to BGP anomalies. It also implies a better match between agent activities and the administrative boundaries. Comparing Figures 4-4, 4-5 and 4-7, 4-8, we can see that the latency and the AS-path lengths correlate better in the synthetic topology than in the real topology.

Figures 4-6 and 4-9 show the percentage of probing traffic in the hybrid PNS schemes, compared with PNS. The number of probing message in PNS is considered to be 100%. We can see that a proper filtering function can significantly reduce network traffic, and the traffic decreases with the network size. For example, on the real topology with a network size of 1,000 nodes (Figure 4-9), $Best$ only require about 20% of probing messages of PNS, and it decreases as the network size increases, finally reaches 12% when the network size is 10,000 nodes. In contrast, $Looser$ requires about 75% of probing messages of PNS on average. $NoProbe$ does not require any probing; hence it falls on the x-axis. The percentage of probing traffic goes down as the network size increases, because as the network size gets bigger, there are more candidates for each entry, and the filtering is more effective with more candidates. Comparing Figure 4-6 and 4-9, the percentage of messages in Figure 4-6 is much lower than that in Figure 4-9.

To evaluate the gains we obtain from saving probing traffic, we need to know the percentage of traffic that is generated by probing messages in a network, compared with the total amount of control messages. Control messages consist of all messages that are used to maintain the cross-region organization and routing correctness, and traffic generated by lookup and data transfer is not included here. This percentage depends on several factors, including the churn rate, the frequency of routing table entry exchange, the number of en-

tries exchanged, leaf set size, the frequency of keep-alive messages, etc. As mentioned before, our simulation is configured in the same way as Pastry with keep-alive messages for leaf sets and lazy routing table repairs [5]. Under such configuration and a network size of 10,000 nodes, we observe that the number of probing messages accounts for about 40% of the total number of messages in the PNS scheme. In Figure 4-9, when the network size is 10,000, probing messages in $Best$ is about 12% of that in PNS. Therefore, if all the other messages are the same, the total number of messages in the $Best$ scheme is about 64.8% of that in the PNS scheme.

Comparing Figures 4-4 to 4-9, we can see that the hybrid PNS schemes perform better on the synthetic topology than on the real topology in both reducing lookup latency and saving probing traffic. The reason for this is that the synthetic topology is more homogeneous than the real one. In the synthetic topology, each AS contains the same number of nodes, and the latency on a link within an AS is set to be smaller than that between two ASes. In contrast, the real Internet is more heterogeneous in both the number of nodes in an AS and latency distribution. In the real topology, several big ASes may contain many nodes. As a result, a node in such an AS cannot tell the difference between its neighboring nodes just based on AS-path lengths, so the filtering does not work as well as that on the synthetic topology where nodes are uniformly distributed in ASes. The latency distribution is not as uniform as that in the synthetic topology either. But we can see that even on the real Internet data, the hybrid PNS schemes still work quite well and save a large percentage of probing messages. For example, $Best$ can use 12% probing messages to achieve lookup latency only 20% longer than PNS.

The simulation results show that even with some simple filtering functions, we can achieve similar lookup performance to the standard PNS, with a much reduced number of probing messages. The results also show that there is a tradeoff between improving lookup performance and reducing probing traffic. The more we probe, the better performance we can obtain. Different nodes may also adopt different filtering criteria. For example, if a node is in a backbone AS, then it should uses a filtering function that does not distinguish between 0 and 1 AS hop, because a node in the same AS that is far away from the present node will probably have a long latency. Furthermore, filtering functions provide a node

the opportunity to control the amount of probing. A peer can choose a filtering function to balance its requirement on lookup performance and the bandwidth consumption.

Currently we only use very simple and coarse-grained AS-path lengths. With more fine-grained information, we can expect the filtering functions based on such information will achieve better performance. For example, if we know the router-level hops or the structure of points of presence (PoPs) in ASes, we can make more accurate estimate on network latency using this static information. But this requires more efforts to retrieve the underlying network information. Again, there is a tradeoff between cost and benefit.

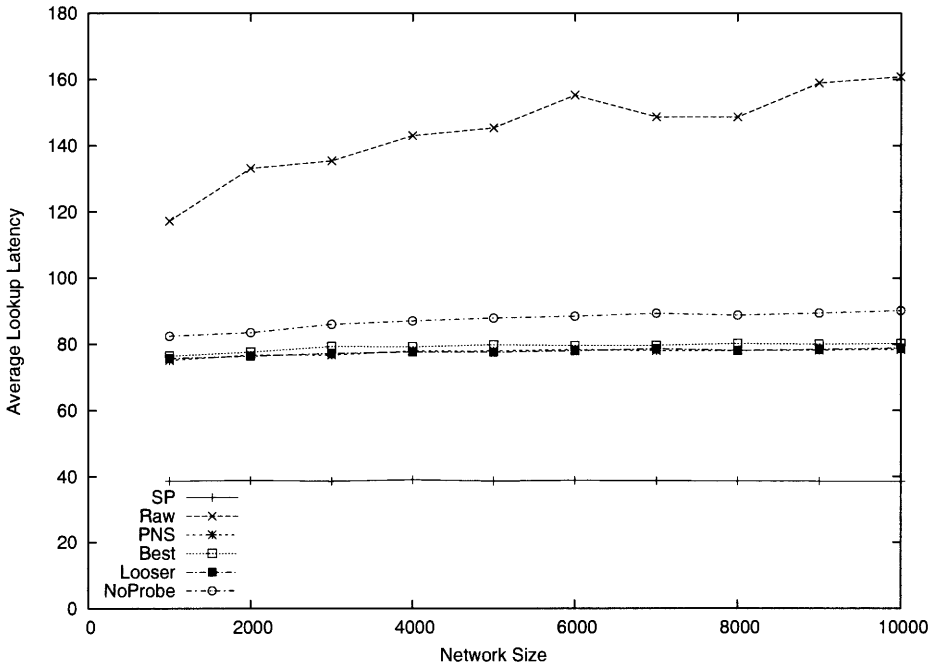


Figure 4-4: Synthetic topology: Average lookup latency.

4.6 Discussion

With the prosperity of overlay networks, the interaction between overlays and ISPs becomes an important issue [10]. To form a synergistic coexistence, overlays and ISPs must coordinate their interests and behavior. Our preliminary results have showed that AS information can help to match overlay activities (agent activities in this work) with the ISP policies.

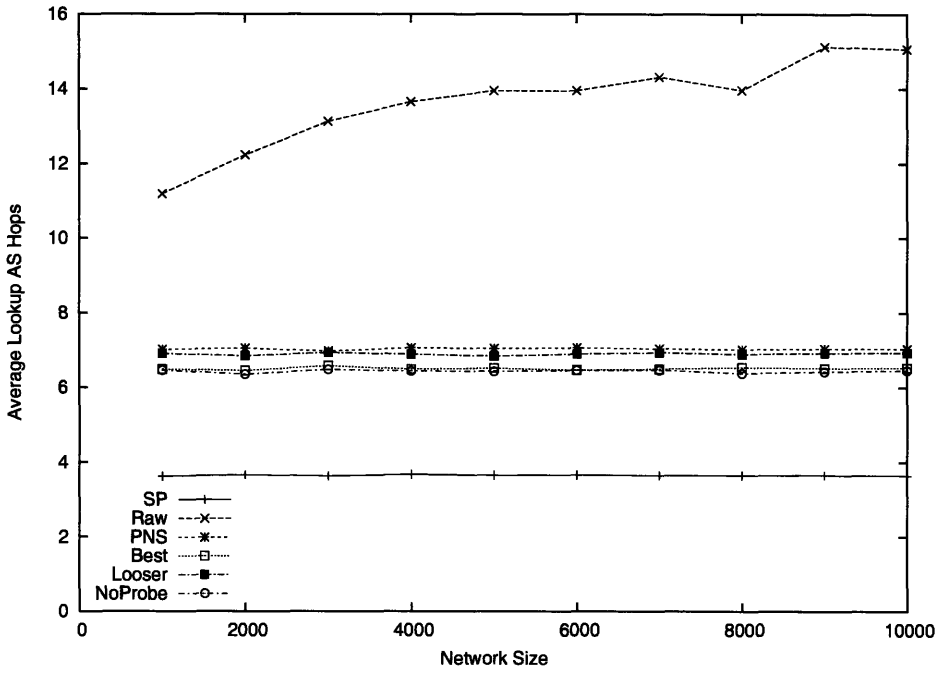


Figure 4-5: Synthetic topology: Average lookup AS hops.

Simulation results show that our schemes can reduce the traffic while maintaining comparable lookup performance. These are attractive features for overlays and end users, who therefore would probably use such techniques in network applications.

Furthermore, we also show in the performance evaluation that our methods achieve shorter AS hops in the lookup. This implies that our schemes naturally shift overlay activities with respect to administrative boundaries. Therefore, by using our techniques, users tend to keep overlay traffic within nearby ASes. This change in traffic pattern is beneficial to ISPs, as such traffic crosses fewer ASes and thus may cost ISPs less.

We can push further to match overlay/agent activities with ASes. A straightforward method is to fill the routing table with nodes in the same AS whenever possible. As a result, most lookup hops will be restricted within the local AS. Furthermore, we need to consider the data flow between the requesting node and the replica. Our AS-based client clustering uses the mapping from IP address to their AS number, and places an additional replica in the AS from which many requests come. Therefore, future requests from the same AS will find the replica in their local AS, and the downloading activities are within the administrative boundary.

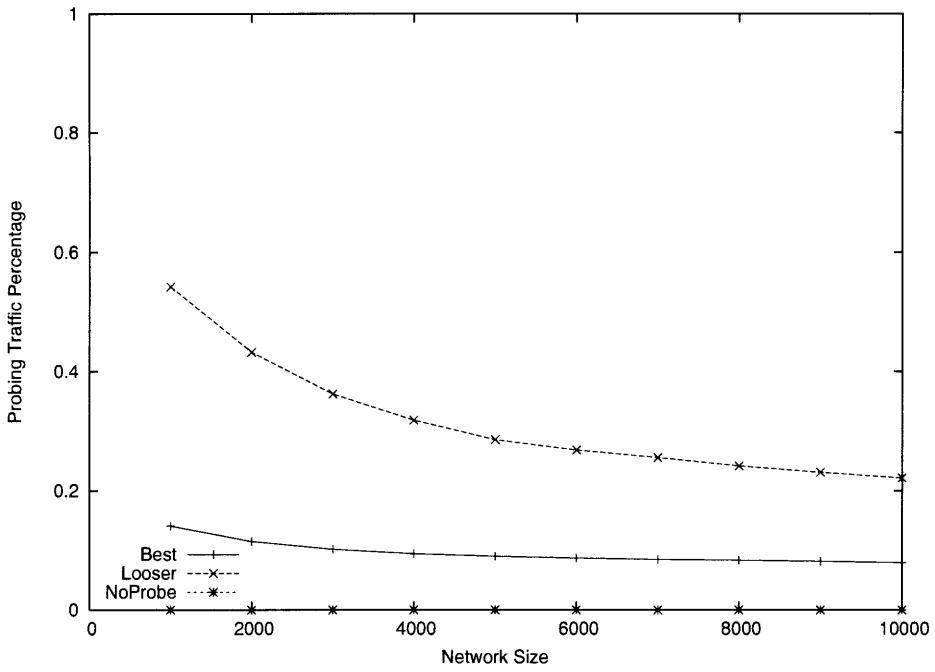


Figure 4-6: Synthetic topology: Probing traffic percentage.

In return, such traffic pattern will motivate ISPs to expose more AS-level routing and topology information to overlays and applications willingly. In this way, we hope that overlays and ISPs can become more aware of each other and learn to coexist peacefully.

4.7 Related Work

This chapter focuses on achieving efficiency and non-intrusiveness in the cross-region organization. In DHTs, many lookup optimizations on have been proposed, such as proximity neighbor selection, proximity routing, topology-based ID assignment and hierarchical routing, but they all require active probing of the network [54], as we explain below.

Pastry employs proximity neighbor selection to improve lookup performance [109]. The idea is to build a topology-aware P2P network. In this technique, each peer sets up its routing table using nearby nodes that satisfy the entry requirement in ID prefix. The effect of this method depends on the flexibility of each routing table entry in choosing nodes, i.e., the length of the prefix. As shown in Table 4.1, the higher rows of the routing table require shorter prefix, thus have more flexibility in choosing nodes. This method does not affect

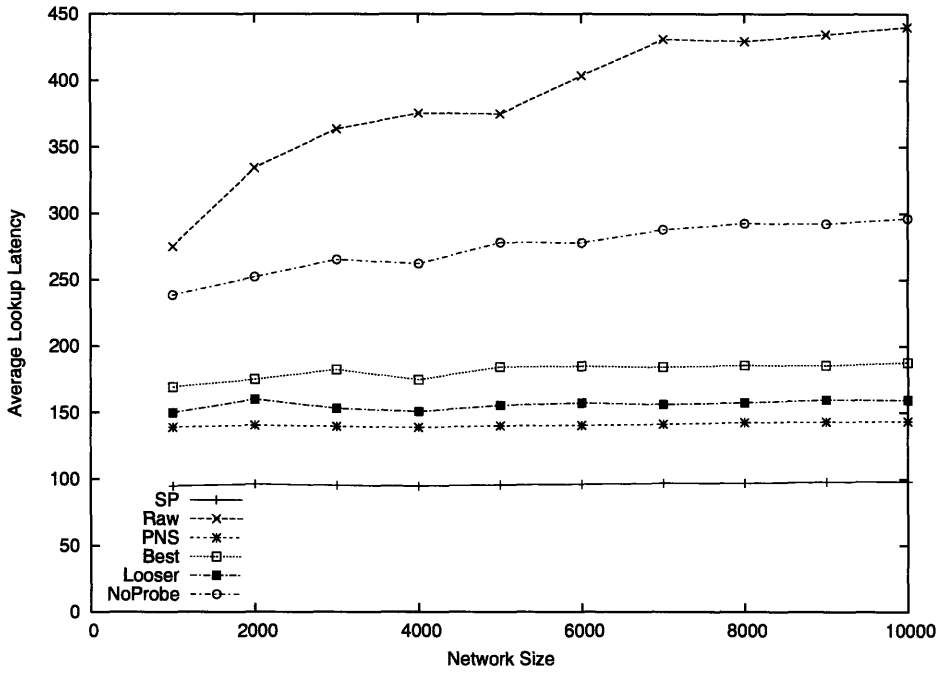


Figure 4-7: Real topology: Average lookup latency.

the total number of hops in a lookup. This technique can also be applied to Chord with minor modifications by allowing each finger to point not to a specific position but one in a range in the name space.

Chord uses proximity routing [122]. Unlike proximity neighbor selection, proximity routing does not require each node to set up its finger table using nearby peers. Proximity routing happens during a lookup process. At each step of a lookup, there may be multiple next hops that are closer to the destination ID than the current node in the name space. Therefore, the current node can choose the closet next hop in physical networks among all possible ones, or one that is a good tradeoff between the physical distance and the progress in the name space. This technique improves the lookup performance. However, it is limited by the number of available next hops. Furthermore, unlike proximity neighbor selection, always choosing the shortest next hop may lead to an increase in the total number of hops, which affects the lookup performance.

Ratnasamy et al. introduces landmarks as topology signals in CAN [106]. With landmarks, a node can map its logical ID to its location in the physical networks, so that neighbors of a node in the overlay network are close to it in the physical network. In this way,

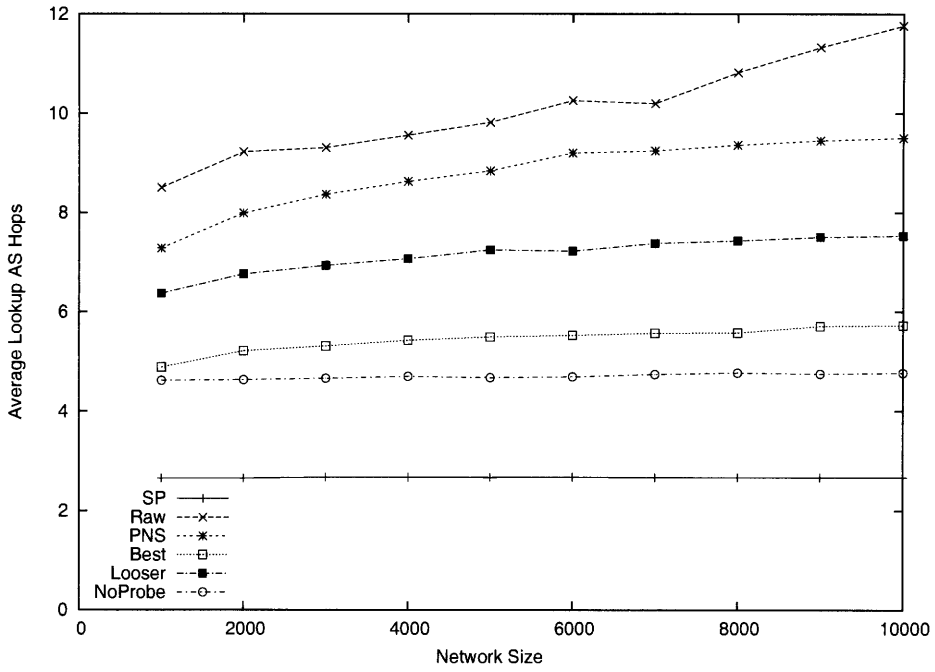


Figure 4-8: Real topology: Average lookup AS hops.

a lookup can reach the destination without going through some faraway nodes. But such topology-based ID assignments violate the uniform distribution of node IDs, which leads to load-balancing problems. Neighboring nodes are also likely to suffer correlated failures. Such a technique cannot be applied to those using one-dimensional name space such as Chord and Pastry.

Brocade adds a secondary overlay on top of the P2P network [134]. The secondary layer consists of super nodes, which are nodes near the network access points such as routers. Each super node manages a set of local nodes, and network traffic is reduced. But super nodes may become the bottleneck, and thus the system does not scale very well.

Xu et al. propose to build Expressway [129]. Expressway is an auxiliary network on top of the structured P2P network, based on AS-level topology derived from BGP tables. An Expressway network consists of nodes near network access points that have good network capacity. It uses routing advertisement similar to the distant-vector algorithm to improve routing performance in the Expressway. None-expressway nodes use their local expressway nodes to route messages. This approach requires that each node know all nodes in the same AS. Furthermore, the cost of distant-vector routing advertisement under a dynamic

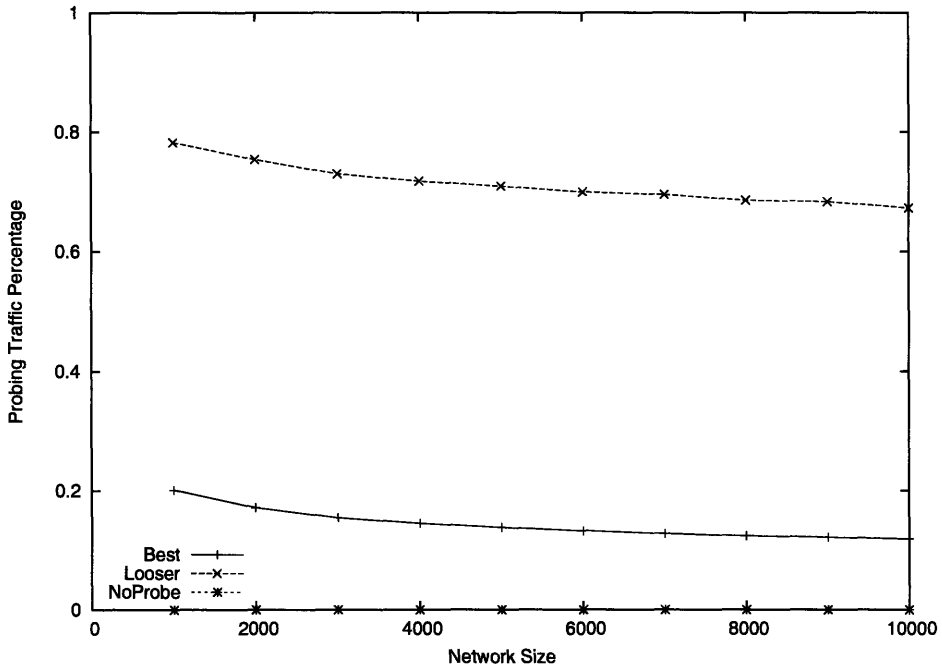


Figure 4-9: Real topology: Probing traffic percentage.

environment is not clear.

Another category of related work is synthetic coordinate systems [87, 38]. Such systems allow hosts to predict end-to-end latencies using synthetic coordinates without probing each other, which can be used by the P2P lookup to avoid contacting distant hosts. GNP [87] relies on a set of landmarks, and each host computes its coordinates using the latencies between itself and all the landmarks. Vivaldi does not require fixed landmarks [38]. In Vivaldi, a host can compute its coordinates after collecting latency information from a few other hosts. Compared with those systems, our approach based on AS-path length is less accurate, but can help to match P2P activities with ISPs.

4.8 Summary

In this chapter we have examined and evaluated algorithms for the leader organization between regions using network topology knowledge and distributed hash tables. This contrasts with the current approaches used in DHTs which are generally based on the discovery of latencies between peers by active probing. We demonstrate that there are significant

advantages to our approach. The first is the significantly reduced probing traffic and performance improvement. Using our hybrid proximity neighbor selection scheme with the AS hop method, we can achieve nearly the same lookup performance as the standard proximity neighbor selection with only 9% of probing messages on the synthetic topology, and 16% longer average lookup latency with only 12% of probing messages on the real topology.

Perhaps even more importantly, the AS infrastructure reflects administrative boundaries in the Internet. Algorithms such as those presented in this chapter allow for restricting agent activities to either individual or sets of ASes that reflect such boundaries. We conclude that with more study, one can generalize these ideas more broadly to the Internet layered architecture, with significant performance and policy benefit.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

Aggregation and Broadcast

Aggregation and broadcast are two important functions for leaders in the region organization. First, as discussed in Chapter 3, not all requests can be resolved by following network topology, and some need to be broadcast to all the regions. Second, similar requests or knowledge can be aggregated to scale to the Internet scope. Third, a new agent may need to broadcast in the NetKP to find a spec-KP that matches its interest or to find relevant agents to form a new spec-KP. Finally, the broadcast and aggregation functionality is needed in many spec-KPs. Therefore, the NetKP should support efficient broadcast and aggregation mechanisms at the region level. In this chapter, I present an aggregation and broadcast mechanism based on the tree structure, and design the tree maintenance protocol.

In Chapter 4, I have presented a network topology based distributed hash table for cross-region organization. DHTs represent an effective way to share information, since there are no central points of failure or bottleneck. However, the flip side to their distributed and dynamic nature is that it is not trivial to aggregate and broadcast global information efficiently. As aggregation and broadcast are fundamental services in the NetKP, we design a novel algorithm for this purpose in this chapter. Specifically, we propose to build an aggregation/broadcast tree in a bottom-up fashion by mapping nodes to their parents in the tree with a *parent function*. This approach also allows us to construct multiple interior-node-disjoint trees, thus preventing single points of failure commonly in the tree structures. In this way, we provide regions with an ability to collect and disseminate knowledge efficiently on a global scale.

5.1 Introduction

In this chapter, our goal is *to design and implement aggregation and broadcast functionalities among regions.* We believe that a good aggregation and broadcast scheme for the regions must satisfy three criteria: accuracy, scalability, and robustness. In terms of accuracy, we want the scheme to be able to provide aggregate information with a certain accuracy in a dynamic environment (where nodes and regions join and leave from time to time). With respect to scalability, we want to minimize message passing and avoid flooding schemes that generate excessive redundant messages, to make the scheme scalable to a large network. We also want to ensure that there is good load-balancing, in the sense that no node among the leaders should be responsible for forwarding a disproportionate amount of network traffic. In terms of robustness, a scheme should be resilient to the dynamics of nodes joining, leaving, and failing arbitrarily among leaders.

The tree is a natural and efficient structure for aggregation and broadcast for the following reasons. To broadcast information, the root can send the information to all its immediate children, and then each child broadcasts in its subtree iteratively, until leaves are reached. During this process, each node only receives the information once, so there is no redundant transmission. When the tree is properly designed, each node has approximately the same number of children, and thus the workload is evenly distributed among nodes and the process completes in the logarithmic number of steps in terms of the tree size. The same argument applies when the information is supposed to arrive at a sink.

Therefore, we propose to build and maintain an efficient and robust tree structure among the regional leaders. In the Chapter 4, I have presented an efficient distributed hash tables using network topology knowledge, and thus we have at hand an efficient DHT-based cross-region organization with a circular and continuous name space. With this underlying infrastructure, we propose a bottom-up approach that constructs and maintains the tree using soft-state, in which a node dynamically determines its parent using a *parent function.* The particular parent function family we provide allows the efficient construction of multiple interior-node-disjoint trees, thus preventing single points of failure in tree structures, and also distributes the aggregation and broadcast load evenly among nodes. With such an

aggregation/broadcast structure, our approach only generates a small amount of network traffic compared with those based on gossiping or flooding, and is robust to node failures.

The rest of the chapter is organized as follows. In Section 5.2, I first give an overview of cross-region organization, and define parent functions to facilitate our discussion. Then I present a bottom-up tree construction and maintenance algorithm, and analyze aggregate accuracy under node failures. Section 5.3 discusses desirable properties that a good parent function should have, gives a sample parent function (family), and examines its features. We also look at how parent functions provide a convenient way to construct multiple interior-node-disjoint trees, thus preventing single points of failure. Section 5.4 discusses implementation issues. Section 5.5 presents the simulation results on the properties and performance of our scheme. Related work is discussed in Section 5.6. Section 5.7 concludes the chapter and highlights current areas of ongoing work.

5.2 A Bottom-up Tree Building Algorithm

In this section, we first give an overview of the cross-region organization, and define parent functions. Then we describe bottom-up tree construction and maintenance protocols in detail, and discuss the advantages of such protocols. Finally, we analyze the aggregation/broadcast accuracy in case of node failures.

5.2.1 Cross-Region Organization Overview

Because the organization of regional leaders from Chapter 4 is based on a distributed hash table, it inherits three key features, which are the base of the design. First, the name space of the regional leaders is circular and continuous, although some are one-dimensional, and some are multiple-dimensional such as CAN. Second, this organization provides efficient lookups. It can resolve a lookup in $O(\log n)$ or fewer steps, where n is the number of the leaders/regions. Finally, this structure is resilient to node failures in that they automatically repair the organization when nodes leave or fail.

With these three features (continuous name space, efficient lookup, and robustness), our goal is to build an efficient and robust aggregation/broadcast tree among regional lead-

ers. As nodes are often used to describe the entities in DHTs, and we focus on the DHT formed by regional leaders in this chapter, we make nodes and leaders exchangeable in the following discussion.

5.2.2 Parent Function Definition

The key idea in our bottom-up tree-building algorithm is to use a many-to-one function, $P(x)$, to map each node uniquely to a parent node in the tree based on its id x. More specifically, the parent node for a node x is the node which owns the id $P(x)$. If node x owns $P^i(x)$ for $i = 1, \cdots, \infty$, then x is the root of the tree. Note that $P^i(x)$ refers to function iteration of $P(x)$. If we consider nodes in a DHT as nodes in a graph and the child-parent relations determined by $P(x)$ to be directed edges, the resulting graph is a directed tree converging at the root. A tree example on the DHT name space is shown in Figure 5-1. In the following we first define the parent function, and will present a parent function example in Section 5.3.2.

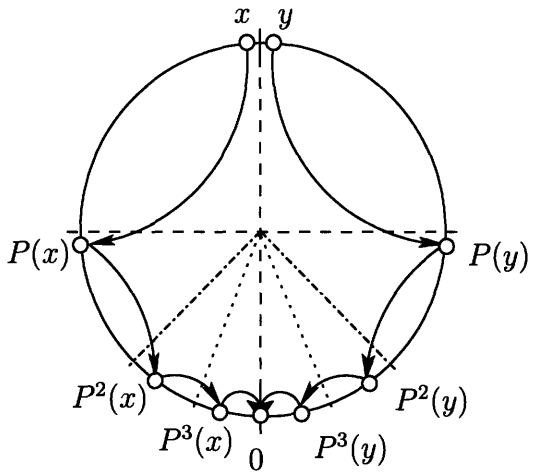


Figure 5-1: A tree example. The circle represents the name space, 0 is the root, and the arrows show the directed edges of the tree from a child to its parent. In this figure, node $P(x)$ is node x's parent, node $P^2(x)$ is $P(x)$'s parent, and so on.

Definition 1. *A Parent Function, $P(x)$, is a function that satisfies the following conditions:*

$$P(\alpha) = \alpha \tag{5.1}$$

$$P^{\infty}(x) = \alpha,\ \forall\ x \tag{5.2}$$

$$Distance(P^{i+1}(x), \alpha) < Distance(P^{i}(x), \alpha),\ \forall\ i > 0, P^{i}(x) \neq \alpha \tag{5.3}$$

where α is an id owned by the root of the tree, x is any valid node id, and $Distance(x, \alpha)$ is the logical distance between id x and the root α, which is essentially the level of id x in the tree.

The above three conditions guarantee that all nodes using $P(x)$ will converge to the root in a finite number of steps, and we prove it in the following theorems.

Theorem 1. *If a function $P(x)$ satisfies the above conditions, there is a directed path from all nodes to the node that owns the id α, which is the root.*

Proof. Theorem 1 is a direct consequence of *Condition (5.2)*. Given an arbitrary node x, we know that there is a path from x to $P^{i}(x)$, $\forall\ i > 0$. Since $P^{\infty}(x) = \alpha$, there must be a path from x to α. □

Theorem 2. *If a function $P(x)$ satisfies the above conditions, all nodes will converge to the root node in a finite number of steps in a finite network.*

Proof. From *Condition (5.3)*, we know that a node that owns $P^{i}(x)$ ($\forall\ i > 0$) is closer to the root α than its child x in terms of $distance$ defined in the parent function. Since there are a finite number of nodes, a node will converge to the root in a finite number of steps, if there is no loop (either back to itself or to a node that is even further away from the parent). We prove that it is impossible to have a loop by contradiction. Without loss of generality, suppose there is a loop in which $P^{k}(x) = x$ ($k > 0, x \neq \alpha$). It implies that $Distance(P^{k}(x), \alpha) = Distance(x, \alpha)$. This contradicts *Condition (5.3)*. □

Note that the name space of the leaders, like that of DHTs, is much larger than the number of leaders/regions. For example, Chord [122] uses a 160-bit name space, which

can theoretically hold up to 2^{160} nodes. In contrast, in the widely-deployed P2P network Gnutella, only about $40,000$ peers were observed at a time and $1.2M$ nodes over an 8-day period [112], which is only a small fraction of the name space. Therefore, a node in a DHT is usually responsible for a range of *id*s, instead of its own *id* only. Like in Chord [122] and many other DHTs, we assume a node is responsible for the *id*s or keys between its predecessor (exclusively) and itself (inclusively), and the node is called the *id*'s *successor* or *owner*. Accordingly, we do not require that a node with an exact *id* of $P(x)$ exists. Instead, as long as a node is currently responsible for *id* $P(x)$ according to the underlying DHT, the node represents the *id* $P(x)$. This rule also applies to the root α such that any node that is currently responsible for *id* α is the root, but usually the root of a tree is the node that initializes this tree and should be alive through the aggregation/broadcast process.

5.2.3 Tree Construction Protocol

With a parent function $P(x)$, we can construct an aggregation/broadcast tree by having each node determine its parent node. The tree construction protocol describes how a node joins an existing tree as follows, illustrated in Figure 5-2.

1. When a new leader, say x, joins the system, we assume that it should already know some existing leaders from which it can set up its state. It is also from the introducing nodes that node x learns about the parent function, $P(x)$, usually parameterized by the root *id* α.

2. After node x joins the system, it learns the *id* range that it is responsible for. If α is within this range, node x knows that it has become the new root of the tree. The former root also knows this since node x is its immediate neighbor, and neighbor set maintenance in the cross-region organization guarantees that the former root knows the joining of node x. The former root will take actions according to the tree maintenance protocol in Section 5.2.4.

3. If node x is not the new root, it must find its parent node using $P(x)$. If $P(x)$ falls into its own *id* range, $P^2(x)$ is computed and checked if it is still in its own *id* range.

This continues until $P^i(x)$ is found not in its range. This is guaranteed to end and a $P^i(x)(i > 0)$ will be found in a finite number of steps by *Theorem 2*.

4. Node x then performs a lookup for the $P^i(x)$. The lookup resulting node, say node y, will become x's parent in the tree.

5. After finding node y, node x sends a message containing $P^i(x)$ to y to register itself as y's child.

6. After receiving x's register message, node y will add node x to its list of child nodes together with the received $P^i(x)$. If node y already has too many children to handle, it will use some admission control to redirect node x to other nodes, as described in the tree maintenance protocol in Section 5.2.4.

Note that in step 3, a node x can usually find its parent by computing $P(x)$, but due to the sparsity of the leaders compared with the name space, it is possible that $P(x)$ is covered in its own id range. In such a case, node x needs to compute $P^i(x)(i > 0)$ where i is the minimum number of times that P has to be applied to x so that $P^i(x)$ maps to a node other than x itself. It can always find its parent in this way unless it is the root. A node can easily figure out whether it is the root by checking if the root id is covered in its id range.

A remaining problem is how to set up the first tree among the leaders, because we need to inform all nodes of the parent function and its parameters first. We choose to build and maintain a default tree with a default parent function $P(x)$ at a default root id α when the NetKP is initialized. We can also depend on either applications or some out-of-band methods to flood the parent function when we want to construct a tree.

5.2.4 Tree Maintenance Protocol

We need to maintain an aggregation/broadcast tree carefully in the cross-region organization where leaders may join and leave dynamically, because a single node failure may cause the subtree rooted at the failed node to lose connection with the other parts of the tree. Therefore, it is crucial to maintain a robust tree that can recover quickly from node failures. The key idea is to let each node maintain its link to its parent node independently

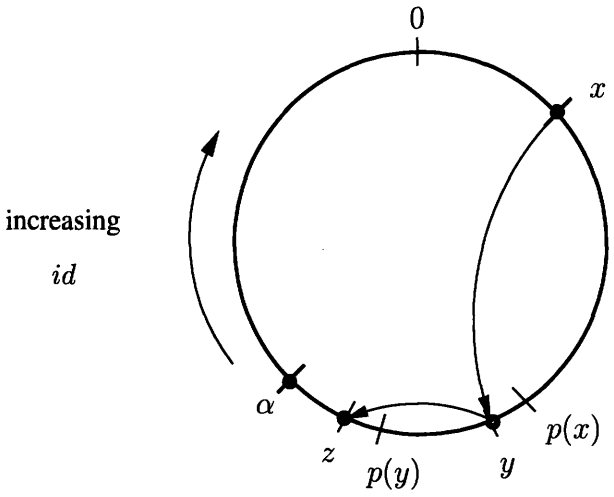


Figure 5-2: The node joining procedure. The circle represents the name space, α is the root *id*, and the arrows show the directed edges of the tree from a child to its parent. In this figure, node y is node x's parent in terms of $P(x)$. Node z is y's parent in terms of $P(y)$.

based on the parent function. Although the parent function gives rise to multiple interior-node-disjoint tree (as will be described in Section 5.3), here we focus on the robustness of a single tree, since this can be easily extended to a variety of multiple-tree maintenance protocol. The tree maintenance protocol is as follows.

1. After a node x joins the tree, i.e., registers at its parent node, it is henceforth x's responsibility to contact its parent node y periodically to refresh its status reliably as a child node. The frequency of the refreshment depends on many factors, such as node failure rate and the requirement of applications. If y fails to hear from x after a specified expiry duration, x will be deleted from y's children list.

2. If node y decides to leave the NetKP, it can notify its children and its parent. To do this, y notifies the children and tells them about its successor which should be their new parent. y will also inform its parent, and its parent will simply delete y from its children list. If y leaves without notification, it is considered as node failure.

3. If node y fails, its child x will detect node y's failure when it tries to refresh its status with y. Then x will perform another joining procedure to find its new parent.

4. Node y will discover that it is no longer responsible for the *id* $P^i(x)$ when a new

node, say z, happens to join between $P^i(x)$ and y, and takes over $P^i(x)$. In such a case, y will inform x that it is no longer its parent in terms of $P^i(x)$ and to its best knowledge, z should be its new parent. After receiving y's message, x will contact z. Based on the received $P^i(x)$, z may either add x to its children list, or tell x that to its best knowledge, another node z' should be x's new parent, if it knows that z' is between $P^i(x)$ and z. This recursive procedure continues until a proper new parent is found. Figure 5-3 shows an example.

5. Node x may notice that it should change the parent. This happens when x's current parent is found in terms of $P^i(x)$ ($\forall\ i > 1$), which implies that $P^j(x), j = 0, 1, ..., i-1$ are covered in x's id range. If x notices that a new node has joined as its neighbor and is responsible for $P^k(x)(0 < k < i)$, x will switch to the new node and simply stop refreshing its status with its former parent. Figure 5-4 shows an example.

6. If the current root node finds that a new node has joined as its neighbor and it happens to cover the root id, the old root knows that the new node will take over the root responsibility, so it will find its parent using the parent function.

7. If a parent node is overloaded because it has too many children, or cannot handle all children due to capability changes, it will ask some children to switch to other nodes. The way for the children to find other parents can be based on the parent function, and we will discuss it in detail in Section 5.3.2.

5.2.5 Discussion

As mentioned before, we maintain a default tree over the leaders. We treat it as a light-weight aggregation/broadcast infrastructure, and as a base to construct other special purpose trees, such as trees for media streaming. To keep it simple, we do not consider factors such as bandwidth or proximity in the default tree. Since the refreshing message is small, we can use it to aggregate and broadcast some general NetKP information. For example, we can piggyback the size of the subtree in refreshing messages, so that the root will obtain the size of all its subtrees and learn the NetKP size. Then the root can piggyback the

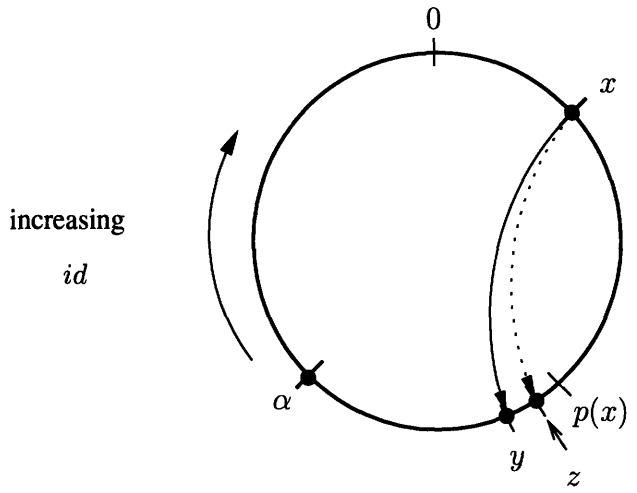


Figure 5-3: The first case of parent change due to node joining. Node y is node x's current parent in terms of $P(x)$. Node z is a new node which joins and covers $P(x)$. Thus z should be x's new parent. y can easily discover this by observing that z becomes its neighbor.

NetKP size in the acknowledgments to its children, so that eventually each node will learn the NetKP size. Other types of aggregate information can also be collected cheaply in the similar way.

There are several advantages of our algorithm over previous tree construction and maintenance schemes. First, our tree based on the parent function is constructed and maintained in a distributed bottom-up way. A parent only needs to passively maintain a list of children without any active detection of their status. Each node is responsible for contacting only one node, i.e., its parent. Therefore, the tree maintenance cost is evenly distributed among all nodes.

Second, a node's parent is determined by the parent function and node distribution in the name space, so each node can find its parent independently. Unlike some previous tree-based broadcast and multicast systems where the tree repair requires coordination of the root or multiple nodes [91, 132, 27], our tree can be repaired simultaneously by each node that detects a failure of its parent. Therefore our tree can be repaired easily and efficiently in case of node failure.

Third, parent changes can be detected and completed easily. As explained in Section 5.2.4, there are two cases when a node should change its parent. Both cases can be detected by either the child or the parent by simply observing its neighbor change, as shown

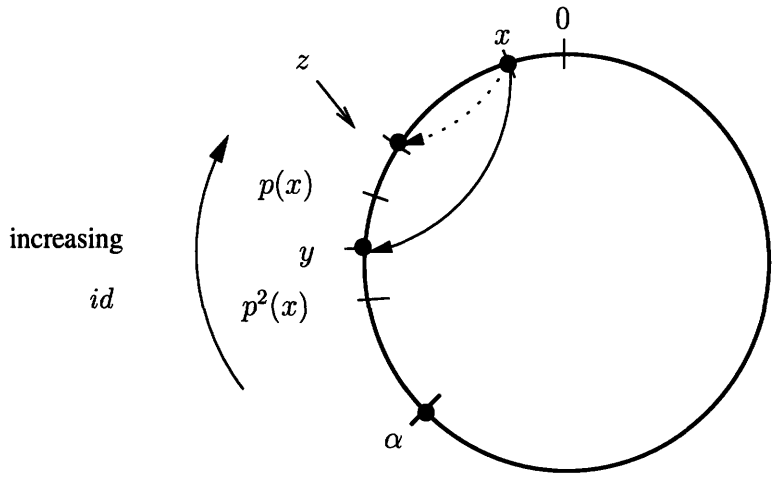


Figure 5-4: The second case of parent change due to node joining. x's current parent is y, in terms of $P^2(x)$, because there are no nodes between x and $P(x)$. Later, z joins and takes over $P(x)$ and thus should be x's new parent. x can easily discover this by observing that z becomes its neighbor.

in Figure 5-3 and 5-4. The first case happens when a new node joins near the parent node. Suppose that a child x registers at its parent y in terms of $P^i(x)$. Later, a new node may join at y's neighborhood and take over $P^i(x)$. It is definitely inefficient to have each node keep detecting whether its corresponding $P^i(x)$ is now covered by another node. Instead, notice that in this case the parent change is triggered by a neighbor change *near the parent node*, and neighbor maintenance already exists in the cross-region organization as in many DHTs. Suppose that y is responsible for an id range $(y_0, y]$. If y observes that some nodes have joined between its predecessor y_0 and itself, it checks whether the new nodes are between $P^i(x)$ and y. If so, it knows that its child x should switch to a new parent. This detection costs nothing since in the cross-region organization each node actively maintains its successor and predecessor. y then notifies x about this change. Note that there may be multiple nodes joining the system simultaneously, so y may not know which new node takes over $P^i(x)$ and the responsibility rests on x itself to disambiguate and find out its new parent. The second case happens when a new node joins *near the child*. Suppose x's current parent is found in terms of $P^i(x)(i > 1)$, which implies that $P^j(x), (0 < j < i)$ are covered in x's own range. Later, if x notices that a new node joins at its neighborhood and takes over $P^k(x)(0 < k < i)$, x knows that it should switch to the new node. Therefore,

tree maintenance cost on parent changes is very low.

5.2.6 Accuracy Analysis

A general aggregation operation consists of three steps. First, a request is sent by a node to the root. Second, the request is broadcast down along the tree to all nodes from the root. Third, all nodes aggregate data of their own subtrees and the data flow up along the tree to the root. Note that in the first step, a request can follow the tree to reach the root, or be sent directly to the root if the requester knows the root, for example, via the parent function presented in Section 5.3.

Although the tree maintenance protocol tries to recover from node failures and maintain a robust tree, in an extremely transient environment where a significant fraction of the nodes can be down within a short period of time, it is very likely that nodes will fail during the broadcast and aggregation procedure. Node failures affect the accuracy in two ways– a node can either fail before forwarding the aggregation request to its children or before forwarding the aggregated result of its subtree to its parent. In both cases, without any recovery mechanism, the final result received by the root will be missing information from the lost subtree. Below we consider the effect of one type of recovery mechanism, namely refreshing, on the conditional probability that the aggregation result is correct, given that the aggregation did occur.

In order to analyze the accuracy of this approach, we make the following assumptions. We divide time into equal intervals of length C_r, and there will be one refresh/probe in each interval. The time that the refresh/probe happens is uniformly distributed within each interval. Therefore, the probability density function (pdf) is constant at $1/C_r$ in each interval. The lifetime of any node follows an exponential distribution with parameter λ_l, as the exponential random variable is a good model for the amount of time until a piece of equipment breaks down or until an accident occurs. The aggregation events are a Poisson process with rate parameter λ_a, as a broadcast or aggregation request can be considered as an arrival and the Poisson process is natural to describe probabilistic arrival events.

Consider the beginning of a refresh interval and label that time 0. Let R be the time

interval from 0 until the next refresh. Let L equal the time interval from 0 until the next failure. Since the lifetime distribution is exponential and thus memoryless, the distribution of L is again exponential with parameter λ_l. Let A equal the time interval from 0 until the next aggregation. Likewise, the inter-aggregation interval is exponentially distributed and thus memoryless, the distribution of A is again exponential with parameter λ_a. For simplicity, we condition on the aggregation occurring at time a, and that all the different characteristics of different nodes are independent, unless stated otherwise.

We consider the conditional probability that the aggregation result is correct from a single node's perspective, i.e. its parent has received its aggregation result.

All the probabilities we talk about below are conditioned by the fact that the aggregation occurred at time a. By the total probability theorem, we can split this into different (sub)cases.

1. *Case 1: Parent does not fail within time 0 to time a.* Conditioned further on this, the probability of aggregating correctly for a node is 1. The probability of this case is $\mathbf{P}[L > A|A = a] = 1 - \int_0^a \lambda_l e^{-\lambda_l l} dl = e^{-\lambda_l a}$.

2. *Case 2: Parent does fail sometime in between time 0 to time a.* The probability that the parent fails sometime in between time 0 to time a is $\mathbf{P}[L \leq A|A = a] = 1 - e^{-\lambda_l a}$. There are two subcases to consider, whether the sequence of events is: failure, refresh, aggregation (*Case 2-1*), or refresh, failure, aggregation (*Case 2-2*). In the latter case the result would be incorrect, so we do not need to calculate it. In the former case, we make another simplification: the refresh always has enough time to complete before the aggregation (so refresh is instantaneous). This can be further divided into two worlds.

   (a) *Case A: $a \leq C_r$.*

   In essence the probability we are dealing with now is
   $\mathbf{P}[\text{Aggregation correct for a node}|A = a, L \leq A, a \leq C_r]$
   $= \mathbf{P}[L \leq R \leq A|A = a, L \leq a, a \leq C_r]$
   $= \mathbf{P}[L \leq R|L, R \leq a \leq C_r] \cdot \mathbf{P}[R \leq A|A = a, L \leq a, a \leq C_r]$

$= \mathbf{P}[L - R \leq 0 | L, R \leq a \leq C_r] \cdot P[R \leq a | a \leq C_r]$

$= \mathbf{P}[L - R \leq 0 | L, R \leq a \leq C_r] \cdot \frac{a}{C_r}$

(again, L, R denotes the lifetime and the refreshing-time r.v.s respectively). Let W denote the random variable $L-R$. We get the following as the pdf of W after doing the convolution $f_W = \int_l f_L(l) f_R(l - w) dl$. Please see the Appendix A for the details of the convolutions.

$$f_W = \begin{cases} 0 & \text{if } w \geq a \\ \int_{l=w}^{a} \frac{1}{a} \frac{1}{1-e^{-\lambda_l a}} \lambda_l e^{-\lambda_l l} dl & \text{if } 0 \leq w < a \\ \int_{l=0}^{a+w} \frac{1}{a} \frac{1}{1-e^{-\lambda_l a}} \lambda_l e^{-\lambda_l l} dl & \text{if } -a \leq w < 0 \\ 0 & \text{if } w < -a \end{cases}$$

Note that in the above $f_L(l)$ and $f_R(l - w)$ are under condition $L \leq a$ and $R \leq a$, respectively. This evaluates to

$$f_W = \begin{cases} 0 & \text{if } w \geq a \\ \frac{1}{a} \frac{1}{1-e^{-\lambda_l a}} (e^{-\lambda_l w} - e^{-\lambda_l a}) & \text{if } 0 \leq w < a \\ \frac{1}{a} \frac{1}{1-e^{-\lambda_l a}} (1 - e^{-\lambda_l (a+w)}) & \text{if } -a \leq w < 0 \\ 0 & \text{if } w < -a \end{cases}$$

Therefore,

$\mathbf{P}[L \leq R \leq A | A = a, L \leq A, a \leq C_r]$

$$\begin{aligned} &= \mathbf{P}[L - R \leq 0 | L, R \leq a \leq C_r] \cdot \frac{a}{C_r} \\ &= \frac{a}{C_r} \int_{w=-a}^{0} \frac{1}{a} \frac{1}{1 - e^{-\lambda_l a}} (1 - e^{-\lambda_l (a+w)}) dw \\ &= \frac{1}{C_r} \frac{1}{1 - e^{-\lambda_l a}} (a - \frac{1}{\lambda_l} + \frac{1}{\lambda_l} e^{-\lambda_l a}) \end{aligned}$$

(b) *Case B:* $a > C_r$.

Here, we are after the same probability

$\mathbf{P}[\text{Aggregation correct for a node} | A = a, L \leq A, a > C_r]$

$= \mathbf{P}[L \leq R \leq A | A = a, L \leq A, a > C_r]$

$$= \mathbf{P}[L \leq R | L, R \leq a, a > C_r] \cdot \mathbf{P}[R \leq A | A = a, a > C_r, L \leq R]$$
$$= \mathbf{P}[L - R \leq 0 | L, R \leq a, a > C_r] \cdot P[R \leq a | a > C_r]$$
$$= \mathbf{P}[L - R \leq 0 | L, R \leq a, a > C_r] \cdot 1$$
$$= \mathbf{P}[L - R \leq 0 | L, R \leq a, a > C_r]$$

Again, we let W denote the random variable $L - R$, and carry out the convolution, and get:

$$f_W = \begin{cases} 0 & \text{if } w \geq a \\ \int_{l=w}^{a} \frac{1}{C_r} \frac{1}{1-e^{-\lambda_l a}} \lambda_l e^{-\lambda_l l} dl & \text{if } a - C_r \leq w < a \\ \int_{l=w}^{C_r+w} \frac{1}{C_r} \frac{1}{1-e^{-\lambda_l a}} \lambda_l e^{-\lambda_l l} dl & \text{if } 0 \leq w < a - C_r \\ \int_{l=0}^{C_r+w} \frac{1}{C_r} \frac{1}{1-e^{-\lambda_l a}} \lambda_l e^{-\lambda_l l} dl & \text{if } -C_r \leq w < 0 \\ 0 & \text{if } w < -C_r \end{cases}$$

Note that in the above $f_L(l)$ and $f_R(l - w)$ are under condition $L \leq a$ and $R \leq a$, respectively. This evaluates to

$$f_W = \begin{cases} 0 & \text{if } w \geq a \\ \frac{1}{C_r} \frac{1}{1-e^{-\lambda_l a}} [e^{-\lambda_l w} - e^{-\lambda_l a}] & \text{if } a - C_r \leq w < a \\ \frac{1}{C_r} \frac{1}{1-e^{-\lambda_l a}} [e^{-\lambda_l w} - e^{-\lambda_l (C_r+w)}] & \text{if } 0 \leq w < a - C_r \\ \frac{1}{C_r} \frac{1}{1-e^{-\lambda_l a}} [1 - e^{-\lambda_l (C_r+w)}] & \text{if } -C_r \leq w < 0 \\ 0 & \text{if } w \leq -a \end{cases}$$

Therefore,

$\mathbf{P}[L \leq R \leq A | A = a, L \leq A, a > C_r]$

$$= \mathbf{P}[L - R \leq 0 | L, R \leq a, a > C_r]$$
$$= \int_{w=-C_r}^{0} \frac{1}{C_r} \frac{1}{1 - e^{-\lambda_l a}} (1 - e^{-\lambda_l (C_r+w)}) dw$$
$$= \frac{1}{C_r} \frac{1}{1 - e^{-\lambda_l a}} (C_r - \frac{1}{\lambda_l} + \frac{1}{\lambda_l} e^{-\lambda_l C_r})$$

We can now calculate the probability of the aggregation being correct from a single

node's perspective, applying independence:

$$
\begin{aligned}
&\mathbf{P}[\text{Aggregation is correct for a node}|\text{Aggregation occurred at } a] \\
&:= \mathbf{P}[\text{correct}|A = a] \\
&= \mathbf{P}[\text{correct}|A = a, L > A] \cdot \mathbf{P}[L > A|A = a] \\
&\quad + \mathbf{P}[\text{correct}|A = a, L \leq A] \cdot \mathbf{P}[L \leq A|A = a] \\
&= 1 \cdot e^{-\lambda_l a} + \mathbf{P}[\text{correct}|A = a, L \leq A] \cdot (1 - e^{-\lambda_l a})
\end{aligned}
$$

At this point, depending on whether we are in case A ($a \leq C_r$) or case B ($a > C_r$), the result is going to be different. Recall that we are fixing $A = a$, so we are either in case A or case B.

In case A ($a \leq C_r$) we have:

$$
\begin{aligned}
\mathbf{P}_A &= \mathbf{P}[\text{correct}|A = a] \\
&= 1 \cdot e^{-\lambda_l a} + \frac{1}{C_r} \frac{1}{1 - e^{-\lambda_l a}} (a - \frac{1}{\lambda_l} + \frac{1}{\lambda_l} e^{-\lambda_l a}) \cdot (1 - e^{-\lambda_l a}) \\
&= e^{-\lambda_l a} + \frac{1}{C_r} (a - \frac{1}{\lambda_l} + \frac{1}{\lambda_l} e^{-\lambda_l a}) \\
&= \frac{1}{C_r} (a - \frac{1}{\lambda_l}) + (1 + \frac{1}{C_r \lambda_l}) e^{-\lambda_l a}
\end{aligned}
$$

and in case B ($a > C_r$) we have:

$$
\begin{aligned}
\mathbf{P}_B &= \mathbf{P}[\text{correct}|A = a] \\
&= e^{-\lambda_l a} + \frac{1}{C_r \lambda_l} e^{-\lambda_l C_r} - \frac{1}{C_r \lambda_l} + 1
\end{aligned}
$$

Note that the inter-arrival time between the aggregation events is exponentially distributed with parameter λ_a. Therefore, we can combine the two cases and get the final formula. Let Z be the event that the aggregation is correct for a node. Table 5.1 shows the

probabilities of several typical settings.

$$
\begin{aligned}
\mathbf{P}[Z] &= \mathbf{P}[\text{Aggregation is correct for a node}] \\
&= \int_0^{C_r} \mathbf{P}_A f_A(a)da + \int_{C_r}^{\infty} \mathbf{P}_B f_A(a)da \\
&= \int_0^{C_r} (\frac{1}{C_r}(a - \frac{1}{\lambda_l}) + (1 + \frac{1}{C_r\lambda_l})e^{-\lambda_l a})\lambda_a e^{-\lambda_a a}da \\
&+ \int_{C_r}^{\infty} (e^{-\lambda_l a} + 1 - \frac{1}{C_r\lambda_l} + \frac{1}{C_r\lambda_l}e^{-\lambda_l C_r})\lambda_a e^{-\lambda_a a}da \\
&= \frac{1}{C_r\lambda_a} - \frac{1}{C_r(\lambda_a + \lambda_l)} + \frac{\lambda_a}{\lambda_a + \lambda_l} - \frac{1}{C_r\lambda_a}e^{-\lambda_a C_r} + \frac{1}{C_r(\lambda_a + \lambda_l)}e^{-(\lambda_a + \lambda_l)C_r}
\end{aligned}
$$

To understand the final result, let us look at several extreme situations:

1. When $\lambda_l \to 0$, P[Z] $\to 1$. $\lambda_l \to 0$ means the node life time goes to infinity. P[Z] approaches 1, because the aggregation will always be correct when there is nearly no node failures.

2. When $\lambda_a \to \infty$, P[Z] $\to 1$. $\lambda_a \to \infty$ means that the inter-aggregation interval goes to 0. In this case, P[Z] approaches 1 because when the aggregation event happens very frequently, the probability that it happens before the node failure approaches 1.

3. When $C_r \to \infty$, P[Z] = $\frac{\lambda_a}{\lambda_a+\lambda_l}$. When C_r goes to infinity, the probability of aggregating correctly is equal to the probability that the aggregation event happens before the node failure event. As the two events are modeled as independent Poisson process, the merged process is a Poisson process with rate $\lambda_a + \lambda_l$, and the probability that the first arrival is an aggregation event is $\frac{\lambda_a}{\lambda_a+\lambda_l}$.

5.3 Parent Function

The protocols in Section 5.2 help to construct and maintain a tree in the cross-region organization, but they do not determine the shape of a tree and other properties: those properties are determined by the parent function. Parent functions play a central role in the properties of an aggregation/broadcast tree. In this section, we discuss the desirable features that

Setting			Probability
C_r	$\frac{1}{\lambda_a}$	$\frac{1}{\lambda_l}$	$Prob$
100	10	5000	99.8%
100	100	5000	98.6%
100	200	5000	96.9%
10	500	10000	95.3%
10	100	1000	91.3%
10	20	100	86.8%

Table 5.1: The probability of aggregation being correct from a single node's perspective under different settings. Note that $\frac{1}{\lambda_l}$ is the average node life time, and $\frac{1}{\lambda_a}$ is the average aggregation interval time.

a parent function should have. Then we present a sample parent function family, show that it satisfies our definition (i.e. satisfies the three required conditions mentioned in Section 5.2.2), and analyze its properties.

5.3.1 Desirable Features

A parent function determines the properties of the constructed tree. We believe that, due to the scale and potential dynamics of the leaders, a good parent function should have the following features to be efficient and flexible, besides the three required conditions mentioned in Section 5.2.2.

First, for the purpose of load balancing, a good parent function should make sure that each parent node has approximately the same number of children, given that nodes are uniformly distributed in the name space. This helps to build a balanced tree. Note that the tree is built on top of the DHT-based cross-region organization, and the assumption of uniform distribution is common in many distributed hash tables, so this assumption does not affect the cross-region organization.

Second, node joining and leaving should not significantly affect the structure of an established tree. We identify that two features of the cross-region organization make it hard to stabilize a tree. One is that, it may sound appealing to have nodes at each level of a tree be evenly distributed in the name space. However, the number of the leaders is

much smaller than its name space, so it is very likely that an id of $P(x)$ will map to the node that follows $P(x)$. A well-intended parent function that makes nodes at each level of a tree evenly distributed will inevitably lead to chaos in a real tree due to the sparsity of the name space. Therefore, a good parent function should guarantee an approximately balanced tree structure under this circumstance. The other feature that makes it difficult to maintain a stable tree is that a node may change its parent node due to nodes joining and leaving. For example, suppose a node x with a large subtree attaches to a parent node y at a higher level. Then a new node z joins as a leaf node in a low level. If x should switch from y to z according to the parent function, then the resulting tree will be very unbalanced.

Third, a good parent function should support branch balancing in a dynamic environment. Although statistically a good parent function can balance the number of children each node has, it is unavoidable that some nodes may be assigned too many children to handle due to the dynamics of the NetKP and variance in node distribution. There are two aspects of branch balancing: admission control and dynamic adaptation. In admission control, a parent decides whether to accept a child when the child tries to register. Admission control is not enough since a parent's condition may change such that it can no longer handle all children, and in such a case, a dynamic adaptation is needed. In both cases, the key problem is how the refused or abandoned children find their new parents. If a node's current parent is overloaded and it has multiple parent candidates, it can switch to another parent.

Fourth, a parent function should allow a child node to pick a parent node optimized according to performance or other metrics. For example, the tree can be optimized according to network topology for performance improvement or according to Autonomous System relationship for economic reasons.

5.3.2 A Parent Function Example

There are many functions that satisfy Conditions (5.1) to (5.3) in Section 5.2.2. The following is an example adopted in this work[1]:

$$P_s(x) = \begin{cases} \alpha + \left\lfloor \frac{(x-\alpha) \pmod m}{k} \right\rfloor \pmod m, \\ \quad \text{for } 0 \leq (x-\alpha) \pmod m < \frac{m}{2} \\ \alpha - \left\lfloor \frac{m-(x-\alpha) \pmod m}{k} \right\rfloor \pmod m, \\ \quad \text{for } \frac{m}{2} \leq (x-\alpha) \pmod m < m \end{cases}$$

where x is the id of the present node, α is the root id, k is a parameter that determines the branching factor of a tree, $k > 1$, and $m = 2^s$, where s is the number of bits for the address space, i.e., m is the size of the name space.

As shown in Figure 5-5, a tree resulting from this function is rooted at a node that owns the id α. The expected branching factor of a spanning tree constructed with this function is approximately k if the nodes are uniformly distributed in the name space (except for the root). The expected height is $O(\log_k n)$, where n is the number of the leaders. Before proving these properties in theorems 4 and 5 respectively, we first show that $P_s(x)$ is indeed a parent function, as per definition 1.

Theorem 3. *$P_s(x)$ is a parent function.*

Proof. This follows from lemmas 1, 2, and 3. □

Lemma 1. *$P_s(x)$ satisfies the first condition of definition 1, i.e. $P_s(\alpha) = \alpha$.*

Proof. Direct substitution yields the desired result. □

Lemma 2. *$P_s(x)$ satisfies the second condition of definition 1, i.e. $P_s^{\infty}(x) = \alpha$, $\forall\, x$.*

Proof. The essence of this parent function is that after each iteration, the distance between the root and the current id $P_s(x)$ is reduced by branching factor k. To make it simple, we make the following substitution.

[1]Note that the modulo operation on negative numbers in $P_s(x)$ is defined as follows: $a \bmod b = b - (-a) \bmod b$, if $-b < a < 0$.

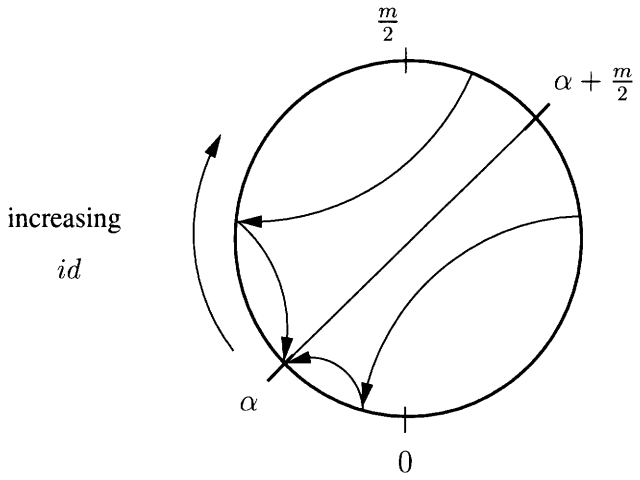


Figure 5-5: Aggregation pattern of the sample parent function. The circle represents the name space, α is the root id, m is the size of the name space, and the arrows show the directed edges from a child to its parent in the tree.

Let d_i be the distance between the current id and the root α after i iterations. We can convert the parent function as follows.

1. When $0 \leq (x - \alpha) \pmod{m} < \frac{m}{2}$, we have:

$$\begin{aligned} d_0 &= (x - \alpha) \pmod{m}; \\ d_{i+1} &= \frac{d_i}{k}; \\ \Rightarrow P_s^i(x) &= (\alpha + d_i) \pmod{m}. \end{aligned}$$

   It is easy to see that $d_i = \frac{d_0}{k^i}$. When $i \rightarrow \infty, d_i \rightarrow 0$, and thus $P_s^i(x) \rightarrow \alpha$.

2. Similarly, when $\frac{m}{2} \leq (x - \alpha) \pmod{m} < m$, we have:

$$\begin{aligned} d_0 &= (m - (x - \alpha)) \pmod{m}; \\ d_{i+1} &= \frac{d_i}{k}; \\ \Rightarrow P_s^i(x) &= (\alpha - d_i) \pmod{m}. \end{aligned}$$

   It is easy to see that $d_i = \frac{d_0}{k^i}$. When $i \rightarrow \infty, d_i \rightarrow 0$, and thus $P_s^i(x) \rightarrow \alpha$.

Thus $P_s^i(x) \rightarrow \alpha$ in both cases. □

Lemma 3. *$P_s(x)$ satisfies the third condition of definition 1, i.e. $Distance(P_s^{i+1}(x), \alpha) < Distance(P_s^i(x), \alpha)$, $\forall\, i > 0$, where $Distance(x, \alpha) :=$*

$$\begin{cases} (x - \alpha)\ (mod\ m), & for\ 0 \leq (x - \alpha)\ (mod\ m) < \frac{m}{2} \\ m - (x - \alpha)\ (mod\ m), & for\ \frac{m}{2} \leq (x - \alpha)\ (mod\ m) < m \end{cases}$$

Proof. In the proof of property 2, it is easy to see that $d_{i+1} < d_i$ in both cases, and therefore we have

$$Distance(P_s^{i+1}(x), \alpha) < Distance(P_s^i(x), \alpha),\ \forall\, i > 0. \qquad \square$$

Therefore $P_s(x)$ is indeed a parent function. We now move on to discuss the expected branching factor and the expected height of a tree constructed using $P_s(x)$.

Theorem 4. *If the node distribution in the name space is identical, independent, and uniform, then k is the expected branching factor of the tree constructed with $P_s(x)$.*

Proof. As n nodes are identically, independently and uniformly distributed in a discrete *id* range [0,m-1], we can use the results in Section 2.5 in Gallager's book [47]. Let X be the interval/distance between two neighboring nodes, then:

$$\begin{aligned} P(X > d) &= (\frac{m - 1 - d}{m})^n \\ \mathrm{E}[X] &= \sum_{d=0}^{m-1}(1 - F(d)) = \sum_{d=0}^{m-1} P(X > d) = \sum_{d=0}^{m-1} \frac{d^n}{m^n} \approx \frac{m}{n} \end{aligned}$$

The number of nodes within an interval of length x, as discussed Section 2.5 in the book, is a binomial distribution with probability of success $\frac{x}{m}$. The interval of length x corresponds to an interval of length $\frac{x}{k}$ with regard to the parent function. Let N be the number of children of a non-leaf node, and X be the id range between the parent node and its predecessor. By law of iterated expectation, we have:

$$\mathrm{E}[N] = \mathrm{E}[\mathrm{E}[N|X]] = \mathrm{E}[n \cdot \frac{kX}{m}] = \frac{kn}{m}\mathrm{E}[X] = \frac{nk}{m} \cdot \frac{m}{n} = k$$

Therefore, the expected number of children for a non-leaf node is k. $\square$

Theorem 5. *If the node distribution in the name space is uniform, then $O(\log_k n)$ is the expected height of a tree constructed with $P_s(x)$, where n is the size of the network.*

Proof. With the previous property, we know that each non-leaf node has an expected number of k children. Suppose that N_i is a random variable that denotes the number of nodes at the tree level i. Note that a non-leaf node has an expected k children according to Theorem 4. Let the height of the tree be a random variable H. Then we have the following:

$$\begin{aligned} n &= \sum_{i=0}^{H} N_i \\ \mathrm{E}[N_0] &= 1 \\ \mathrm{E}[N_i] &= \mathrm{E}[\mathrm{E}[N_i|N_{i-1}]] = \mathrm{E}[k \cdot N_{i-1}] = k \cdot \mathrm{E}[N_{i-1}], for\ i > 0 \\ \mathrm{E}[n] &= n = \mathrm{E}[\mathrm{E}[\sum_{i=0}^{H} N_i|H]] = \mathrm{E}[1 + k + k^2 + \cdots + k^H] = \mathrm{E}[\frac{k^{H+1} - 1}{k - 1}] \end{aligned}$$

Then we have the following inequalities based on Jensen's inequality:

$$\begin{aligned} n &= \mathrm{E}[\frac{k^{H+1} - 1}{k - 1}] \\ &> \mathrm{E}[k^H] \\ &\geq k^{\mathrm{E}[H]} \\ \Rightarrow \mathrm{E}[H] &< \log_k n \end{aligned}$$

Therefore, given that n nodes are uniformly distributed in the name space, the expected height of the formed tree is less than $\log_k n$. □

With the above theorems, we now briefly discuss how our sample parent function $P_s(x)$ satisfies the three desirable features we discussed in Section 5.3.1.

First, our parent function builds a balanced tree. According to Theorem 4, each non-leaf node has an expected number of k children. The expected tree height is $O(\log_k n)$, where n is the number of nodes.

Second, the tree constructed from our parent function is resilient to node dynamics, because, according to the proof of Theorem 4, neighboring nodes are at the same level or the adjacent levels in the tree, and such parent changes will usually move children from

a node to another node at the same or an adjacent level of the tree. On the other hand, if a parent function maps nodes' parents evenly in the name space, its tree will tend to be unbalanced.

Third, our parent function does not provide natural support for branch balancing, but can be easily enhanced as follows. In case of overloading, the parent will ask some of the farthest children in the name space to move. An alternative parent candidate will be found by moving stepwise one neighbor of the parent away from the root, repeatedly until an underloaded node is found. This is guaranteed to terminate because a leaf node will eventually be found in the worst case. After a certain time, the moved children will recompute the parent function, and move back to their normal parents. A nice property of our parent function for branch balancing is that this has little impact on the height of the tree, because nodes near each other are in the same level or adjacent levels of the tree and thus those temporary parents are probably in the same level as the original parent. As a result, the convergence time will not be affected significantly, and the root will not receive redundant information from different aggregation paths.

Fourth, tree optimization according to network topology or other metrics can be achieved with a simple extension. Our parent function determines a unique parent for a given node id, and thus does not provide the flexibility for performance optimization. For example, a child and its parent determined by the parent function may be very distant in the network. One extension is again to take advantage of the property that neighboring nodes are at the same level or the adjacent levels. A child node still uses the same parent function, but the parent returns a list of its neighbors, and the child chooses one of them as its parent, for example, the closest one in terms of latency. During the branch balancing process, a parent will ask the most distant children to move first. This allows certain degree of optimization while maintaining the original tree structure.

Overcoming Single Points of Failure

A common problem of using trees for aggregation and broadcast is that trees are inherently vulnerable to single points of failure. The bottom-up approach will experience periodic glitches in a highly dynamic network.

To address this problem and improve the robustness, we can build multiple trees, and compute the same aggregation/broadcast function over them. By distributing the ids of these trees over the name space uniformly and adopting some quorum system or by averaging over the estimates obtained from several trees, we can further improve the robustness of the aggregation/broadcast.

Depending on their features, some parent functions may provide a convenient way to build multiple interior-node-disjoint trees, such as the parent function we proposed above. Here we show that our sample parent function can easily be used to construct disjoint trees as follows. The key is to adjust parameters in a parent function, so that we can get a family of similar parent functions. Trees constructed using that family of parent functions do not overlap in most of their interior nodes. Note that due to the randomness in the node distribution, this approach does not guarantee a complete disjointness, especially at the boundaries of each level in the name space. We only consider the situation with high probability.

Let us consider our parent function. If a tree is rooted at id 0, then according to the parent function, its non-leaf nodes will be mostly in [0, $\frac{m}{2k}$] and [$m - \frac{m}{2k}$, $m - 1$], which is a range of length $\frac{m}{k}$ in the name space. This is because nodes that are most distant from the root 0 are around id $\frac{m}{2}$, and thus their parents are the non-leaf nodes at the lowest level, and the parent ids are around $\frac{m}{2k}$ and $m - \frac{m}{2k}$, respectively, based on the parent function. Nodes in the other areas are all leaf nodes. Generally, under this parent function, non-leaf nodes in a tree rooted at α are mostly in $[\alpha - \frac{m}{2k} (\text{mod } m), \alpha + \frac{m}{2k} (\text{mod } m)]$, and all other nodes are leaf nodes. Therefore, if we choose k similar parent functions and the distance between two neighboring roots is $\frac{m}{k}$, we can construct k interior-node-disjoint trees, because paths from any two roots to a node in the two trees are disjoint. In this way, we can greatly increase the tolerance of node failures and guarantee that with high probability every node will receive the message at least once. Figure 5-6 demonstrates two disjoint trees when $k = 2$.

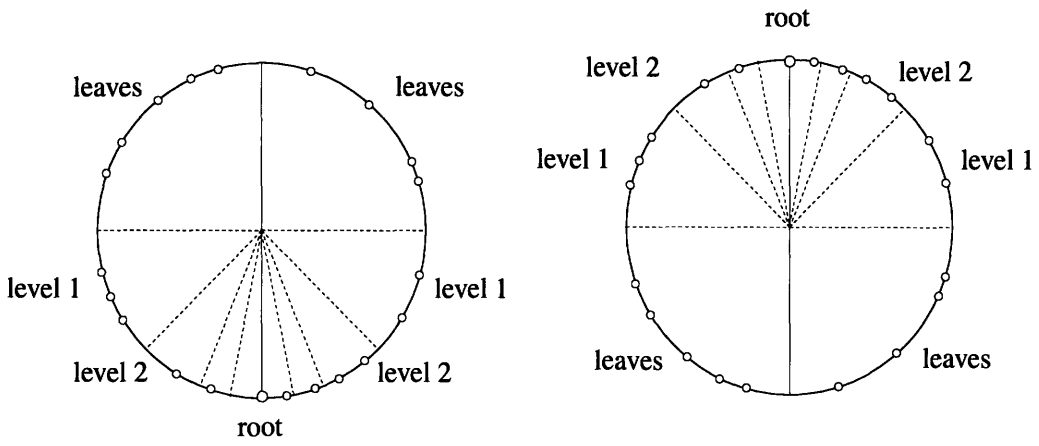


Figure 5-6: Two disjoint trees. Here k is 2, and the two roots are at 0 and $\frac{m}{2}$, respectively.

5.4 Extensions

The construction and maintenance protocols and the parent function help us build a robust tree as a default aggregation/broadcast mechanism. There are several other issues, which are addressed in this section.

5.4.1 Two Operation Modes

As we mentioned before, the tree maintenance messages, i.e., the refreshment and acknowledgment messages, can be used for some light-weight information aggregation and broadcast. For example, we can compute the network size in this way, since it adds little to bandwidth consumption. This is an aggregation/broadcast in the background, which we call the *default mode*.

However, a node may need to perform a special aggregation or broadcast that is useful only for the node itself, for example, searching for files whose names contain certain key words. We call this the *on-demand mode*. The on-demand mode of aggregation consists of the following:

1. When a node wants to perform an aggregation using the tree, it sends a request to the root.

2. Upon receiving a request, the root broadcasts it to its immediate children in the tree,

which in turn forward the request to their children.

3. When a leaf node receives the request, it performs the corresponding aggregation operation, and sends back the result with a time stamp to its parent. If the child does not receive an acknowledgment from its parent within a certain time interval, it determines its new parent and sends the results. Note that it takes some time to find the new parent.

4. A parent node waits for results from all its children. If it does not hear from a child after an expiry time, it will delete this child and not wait for its data. After receiving data from all children, the parent node performs the aggregate operation, attaches a time stamp, and forwards the aggregate result to its parent.

5. If a node receives data from a new child after having sent its aggregate result, it will compute and forward the new result to its parent.

6. If a node receives data from a child several times, but with different timestamps, it will compute the latest result and forward it to its parent.

7. After the root receives results from all children, it will wait for an additional amount of time that is long enough to allow information delayed by parent failures to reach the root. In our implementation, the time equals the height of the tree times the average round trip time. Then the root processes all collected data and computes the final aggregation result. Finally, it sends the result to the request node.

5.4.2 Constructing Multiple Trees

Sometimes we need the capability to set up trees rooted at arbitrary node ids to distribute the load of a root and to provide robustness. Depending on specific applications, we can either rely on the application to disseminate or embed the root identifier, or have one permanent tree rooted at a pre-determined well-known id α_0, as described in Section 5.2.3. We focus on the second case. In this case, if a node, say x, wants to construct a new tree rooted at itself, it will send a message to the root of the default tree, specifying the parent function

with the root id as its own id. The message will then be broadcast down the default tree. All nodes will eventually receive this message, and participate in the new tree rooted at x.

When a new node joins the network, it will get a list of all existing trees from its parent node. Nodes also periodically exchange information on the existing trees. In this way, all nodes will eventually discover and participate in all existing trees.

When the root node of a tree wishes to tear down its tree, it will simply broadcast a message to tell all nodes that the tree should be torn down. If a node does not receive the tear-down message due to its parent failure, it will still get it when it detects the parent failure and switches to a new parent which has already received the message. Therefore, the message is guaranteed to reach all nodes eventually. If a root fails without notification, subsequent messages will end up at a succeeding node which discovers that it has become the root of a tree that it did not set up. It can decide either to keep the tree or to broadcast a message to tear it down.

5.5 Performance Evaluation

In this section, we evaluate the performance of our bottom-up tree-building algorithm using the parent function in Section 5.3.2. As an example of usage, we evaluate the estimation of available network storage under the default mode, and the estimation of the network size under the on-demand mode.

5.5.1 Experiment Setup

Our experiments are divided into discrete time periods. In each period, a number of nodes join the network according to a Poisson distribution with parameter λ_j. At the point when a node joins, its departure time is set according to an exponential distribution with parameter λ_l, and nodes are removed from the network when their lifespans expire. In the default mode, each node sends a refreshment message that contains the aggregate information to its parent in each period. In the on-demand mode, a node refreshes its status under a uniform distribution with parameter C_r, and the aggregation time follows a Poisson distribution

with parameter λ_a. In all experiments, we use the parent function in Section 5.3.2. The root id α is set to 0, and k is 4, unless stated otherwise.

5.5.2 Tree Properties

In the first experiment, we evaluated the overhead of our tree construction and maintenance algorithm in terms of network traffic by counting the number of messages sent in the tree construction process. In this experiment, the node failure rate is approximately λ_l = 10% per time period, and about 10% new nodes join the network too. This keeps the network size roughly stable. During each period, each node refreshes its status with its parent once. The communication cost in terms of messages sent as a function of network size is shown in Figure 5-7.

The total number of messages in each period is at least twice the network size, because each node sends a refreshment message to its parent and receives an acknowledgment (except for the root). Note that refreshment messages can also be used to aggregate and broadcast information in the default mode. Additional messages are needed for new nodes to join, to repair the tree in case of nodes failures, and for overloaded nodes to do branch-balancing. Figure 5-7 shows that the number of additional overhead messages (including tree-repair messages, branch-balancing messages, and node-joining messages) is small, compared with the total number of messages.

An important property is the number of branches or children at non-leaf nodes, as it reflects the overhead of a parent node. Ideally, we would like all the intermediate nodes to have approximately the same number of children.

In the simulation, we define an overloaded node as having more than $2k$, i.e. 8 children. We use the branch balancing algorithm discussed in Section 5.3.2, which moves additional children to its neighbor towards the leaf side. We find that, without branch balancing, there are 3% of overloaded nodes in each period, and the overloading lasts about 3 periods on average. Overloading is automatically resolved when children leave or new nodes join to take over some children from the current parent. The branch balancing procedure usually propagates within 2 neighbors. Figures 5-8 shows the relationship between the network size

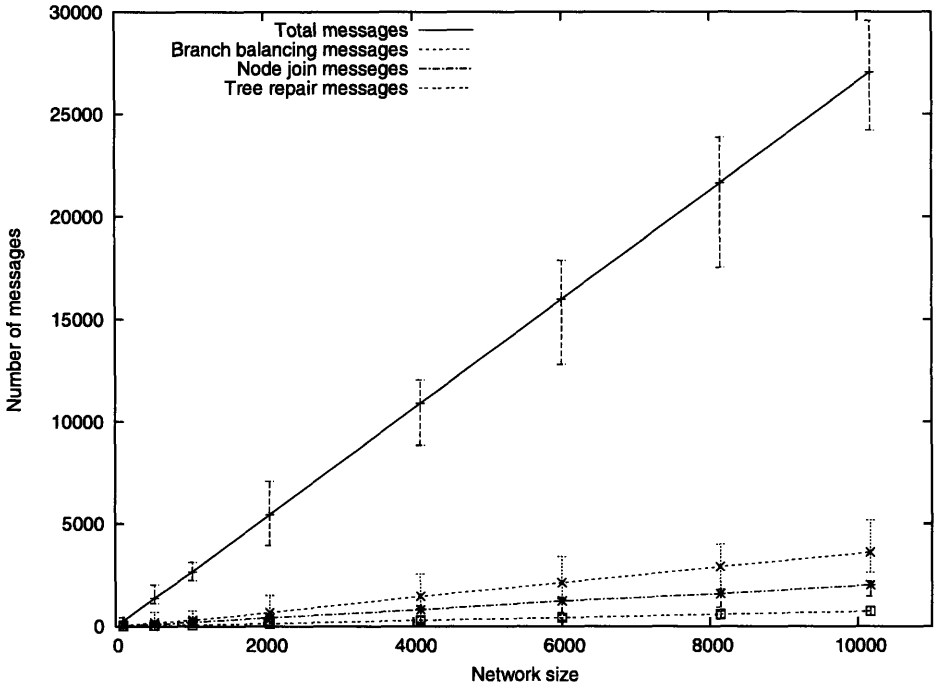


Figure 5-7: Communication overhead for tree construction.

and the branches. We can see that without branch balancing, an overloaded node can take as many as 48 children in a network with about 12800 nodes, and with branch balancing, the maximum branch is only 8, which is the upper bound of the number of children a node can have.

The height of the resulting tree affects the performance since it determines the time it takes for the information to propagate from leaf nodes to the root. Figure 5-9 shows the average tree heights with or without branch balancing under different network sizes. We can see that our branch balancing scheme does not increase the height of the tree, even though the branch balancing affects the tree structure since it does not obey the parent function.

5.5.3 Aggregated Knowledge Estimation

As an example of usage, we use our bottom-up tree to estimate the evolution of the available network storage in the default mode, which aggregates continuously. We estimate the network size in a similar way.

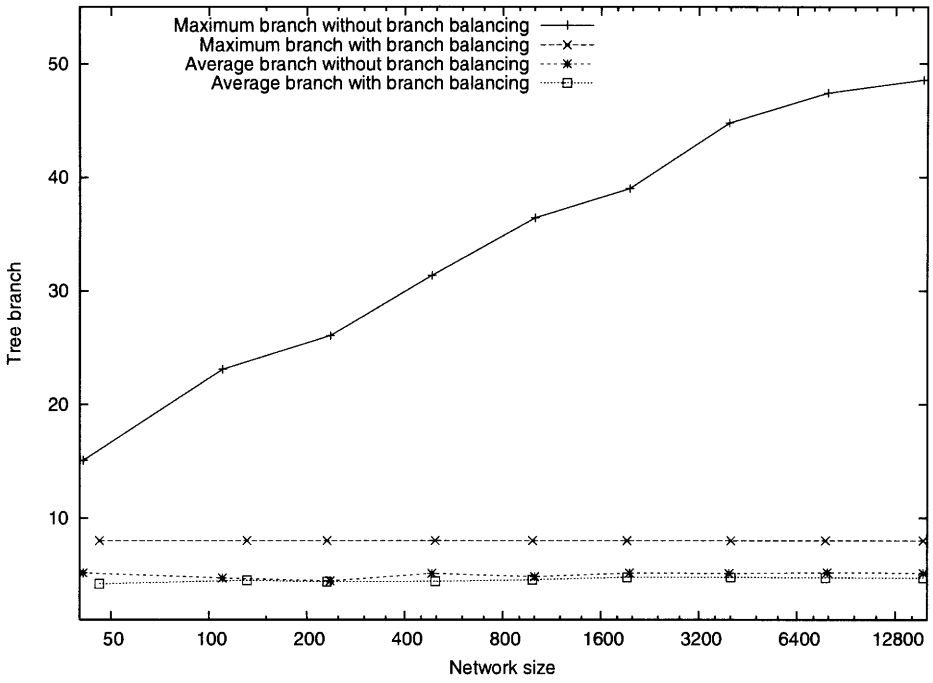


Figure 5-8: Tree branches against network size.

Figure 5-10 shows the evolution of the network storage estimate and the network size estimate at the root of the tree with a node failure rate of $\lambda_l = 5\%$ per time period. The storage on each node keeps changing according to an approximately Gaussian distribution with a mean of 50MB and a standard deviation of 20MB. The simulation consists of three stages: increasing, stable, and decreasing. From Figure 5-10, we can see that the average estimation is very close to the true value. Specifically, in the increasing stage, the estimates tend to be smaller than the actual network storage because there is a lag between the estimate and the actual value. Similarly, during the decreasing stage, the estimates are usually higher than the actual storage size. The spikes are caused by the failure of intermediate nodes high up in the tree, leading to temporary losses or duplicates in the storage information. The results demonstrate that our algorithm is responsive to network storage changes, and recovers rapidly from such failures.

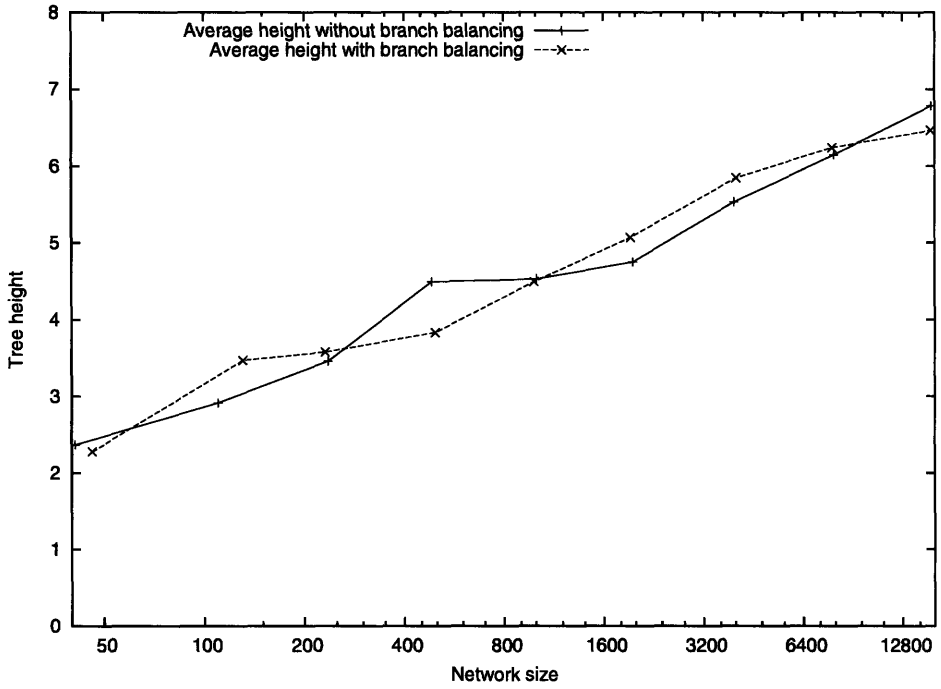


Figure 5-9: Tree heights against network size.

5.5.4 On Demand Estimation

We also evaluate the accuracy of on-demand estimation in the simulation. In this setting, C_r = 10, λ_a = 50, and λ_l = 200. The average estimation over 20 trials is plotted. We set up an initial network of 10000 nodes. During the aggregation, a certain fraction of nodes fail. Figure 5-11 shows the simulation results under various node failure rates. *Before Aggregation* refers to the actual network size before aggregation, and *After Aggregation* refers to the actual network size after aggregation. Since it takes some time for an aggregation procedure to complete, we consider that an estimated value is correct if it is between the original network size and the network size at the end of aggregation. From Figure 5-11 we can see that in most cases the estimation is within the correct range. When node failure rate reaches 15%, some estimates start falling out of the range.

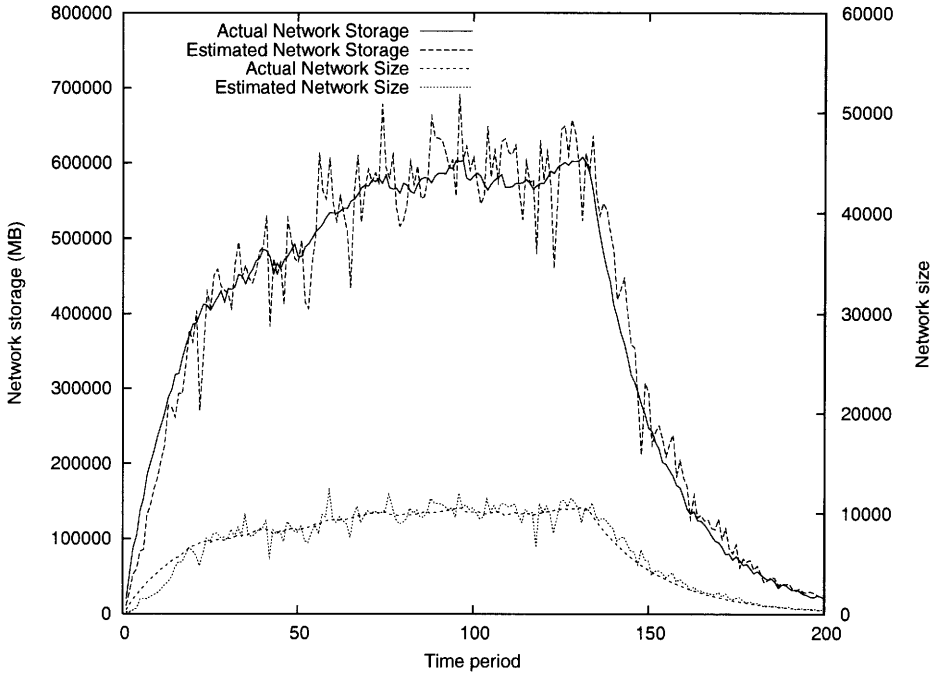


Figure 5-10: Network storage and network size estimation. The solid curve at the top shows the evolution of the actual network storage, and the dashed curve with spikes at the top shows the estimation of network storage. The dashed line at the bottom shows the evolution of network size, and the dotted line with spikes at the bottom shows the estimation of network size.

5.6 Related Work

Many large-scale distributed systems need to collect, aggregate, or broadcast information. For example, in a P2P storage system ([80]) or a Grid-like environment ([92]) it is valuable to learn about aggregate available storage or computation capabilities. In particular, some recently proposed randomized DHT topologies, like Viceroy [77] and Symphony [78], require network size estimates to tune routing performance. Under other circumstances, lifetime distribution and other characteristics may be valuable. Broadcasting can also be used to search for arbitrary semi-structured queries that are not supported by the current DHT lookup. Other applications, such as median distribution, need this functionality, too.

There is a large body of literature in the area of broadcast. In general, existing schemes can be categorized as flooding-based approaches ([72, 97]) and top-down spanning-tree approaches ([44, 29]). A major drawback of the former is that it generates redundant traffic,

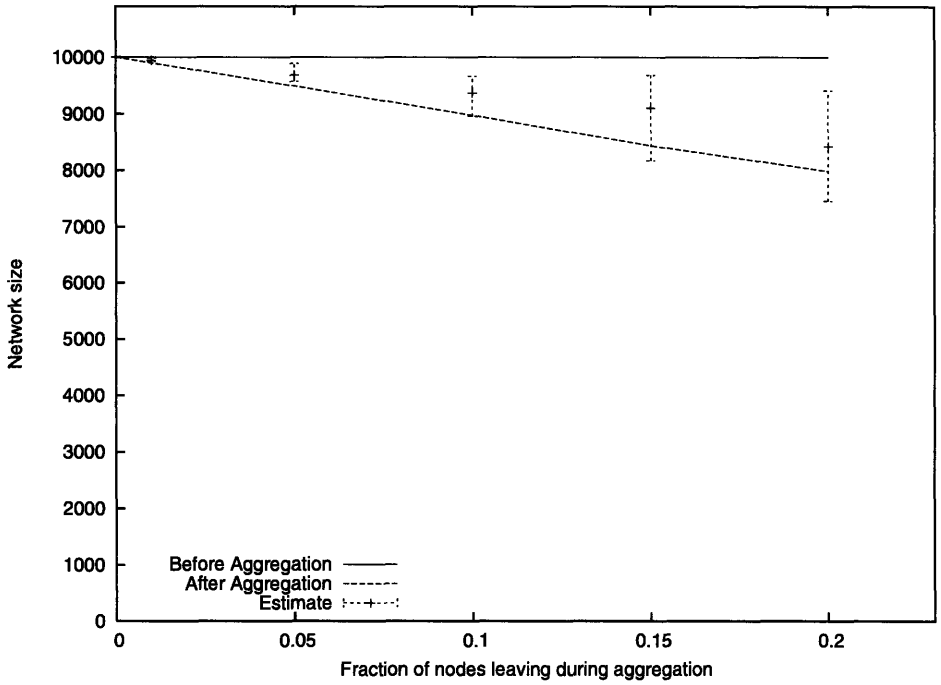


Figure 5-11: On demand estimation under various node failure rates.

while a major drawback of the latter is that when nodes fail in the middle of the hierarchy, large sections of the original spanning tree will lose contact with the root, and reconstruction of the subtrees requires significant efforts. In contrast, we address the problem under more dynamic conditions, and our bottom-up approach not only enables fast recovery from such failures, but also provides an easy way to build multiple trees with disjoint paths so that all nodes can be reached with high probability.

Most of previous overlay multicast systems, such as SCRIBE, Bayeux, etc ([110, 135, 105, 44, 131]) are tightly coupled with underlying overlay networks. For example, in [44], El-Ansary et al. discussed how to use routing tables to construct a broadcast tree, which can be viewed as a special case of our scheme. The advantage of using routing table information is that the child maintenance cost is saved by retrieving children from the routing table, which is always kept updated by the underlying DHT networks. The disadvantage is that the tree structures are constrained by the underlying DHT topologies and thus not flexible. Compared with these systems in which it is hard for a node to adjust the number of its children based on its capability, our bottom-up tree scheme is independent of underlying

overlay and its structure is determined by the parent function, which is flexible in building trees according to different requirements.

In [20], Bawa et al. proposed top-down flooding-based aggregation schemes over unstructured P2P networks, which require a known universal maximum delay and a known upper bound to the network diameter. With the progress in structured P2P networks, we believe that it is feasible to build and maintain a robust infrastructure at relatively low cost without those assumptions, and it is useful to attempt to solve the aggregation problem over the NetKP using a more efficient and scalable approach.

Our bottom-up approach of using a function to map nodes onto parent nodes is similar in spirit to the generalized scheme for building DHTs proposed by Naor and Wieder [84]. In their scheme, functional mapping is used to build the general DHT infrastructure, while in our scheme we seek to build a specific tree structure over the underlying NetKP for aggregation and broadcast. In [132], Zhang et al. built a data overlay independent of the underlying DHT using a top-down method to implement arbitrary data structure, while our tree structure is constructed in a bottom-up fashion and takes advantage of the underlying DHT to maintain the tree.

CoopNet is a distributed streaming media content delivery mechanism [91]. In CoopNet, the root node coordinates all tree management functions, including construction and maintenance. This centralized management makes tree maintenance easy. Trees are constructed incrementally, so the position of a node in the tree depends on when it joins the tree. In our approach, the tree is built in an original bottom-up way. Each node can find its position in the tree without asking any central point, and its position is determined by the parent function. Different from CoopNet, we do not consider bandwidth, since our intention is not for stream media, but for a general light-weight aggregation infrastructure, whose bandwidth requirement is very low.

With respect to dealing with single points of failure, our approach is similar to Splitstream ([27]) in building multiple interior-node-disjoint trees. Splitstream builds multiple interior-node-disjoint trees over SCRIBE [110]. Like most previous work on multicast over P2P networks, trees are built using the routing table entries, and thus the properties and the robustness of the multicast tree relies on the underlying networks. In our approach, the tree

is constructed based on parent functions. Therefore, different applications can build trees based on their requirements using different parent functions. Another advantage of our tree construction is that each node maintains its own parent link and the tree can be repaired simultaneously. In Splitstream, multiple disjoint trees are built based on the base of the id space, while our approach is independent of any underlying routing topology, and thus makes it easier to construct and control the number of trees by simply changing the root id and the branch factor.

5.7 Summary

The aggregation and broadcast for the cross-region organization is complicated by the scale and the dynamics of the NetKP. We propose to construct a robust tree over the regions. We have presented a new, original approach using a bottom-up construction that is based on mapping from a continuous function into the discrete id space. The major advantage is that it has a relatively low overhead and is resilient to node failures. Our scheme is also flexible in that parameters in the parent function can be used to control the tree structure and characteristics, such as the height and the branch factor. We also presented the notion of parent function families to aid in building multiple interior-node-disjoint trees, which helps to alleviate the problem of single points of failure.

Parent functions play an important role in our bottom-up approach. We have only thus far discussed several properties of a parent function that help to improve the tree performance. By adopting a different parent function, we may be able to improve some aspects of the system, or at least achieve some tradeoff between various costs and benefits, which makes our results more widely applicable, since one could conceivably define the cost and benefit function for a particular application and then find a parent function that maximizes the benefit/cost ratio.

Chapter 6

Case Study I: Experiment Management on Testbed

In this chapter, we present a case study to address how different kinds of knowledge are collected, maintained, and used in different ways in an experiment management spec-KP. To demonstrate the effectiveness of this infrastructure, we build experiment management tools on PlanetLab to facilitate the experiment setup by using the knowledge collected in the spec-KP.

6.1 Introduction

Network testbeds have become an important facility for researchers to study large-scale distributed systems. There are several testbeds that are widely used, such as PlanetLab, Emulab, DETER, etc. PlanetLab [96] is an open platform for developing, deploying and accessing planetary-scale services. It is composed of more than 800 nodes at about 400 sites worldwide. At each moment, there are multiple researchers setting up or running their experiments within their own slices. More than one thousand researchers at academic institutions and industrial research labs have used it to develop new technologies for distributed storage, peer-to-peer systems, overlay networks, etc. An alternative is the Emulab-based testbeds that reflect a slightly different approach to experimentation [1]. Among them, the DETER testbed is a public facility of medium-scale repeatable experiments in computer

security [42]. It has been configured and extended to provide effective containment of a variety of computer security experiments. The testbed contains about 300 nodes at two sites, and the nodes are shared among multiple simultaneous experiments isolated from each other. As a result, network testbed and experiment management have become an important and complex issue, as operators need to manage a large number of nodes where many users are running different experiments, some involving security issues [95, 21].

Those testbeds provide different styles of experimentation to the researchers. What we notice is that they have problems in common, including how to set up experiments, how to pick nodes with desired features, how to monitor the experiment to make sure that it runs as specified, etc. In this work, we focus on the PlanetLab environment, as PlanetLab has a large number of nodes widely distributed and thus demonstrates real, complicated network and node behaviors.

In network testbeds, experiment management usually involves the following steps:

1. *Upload the software.* The user uploads the software to all the nodes he chooses. When the number of nodes is large, the user can use tools like *pscp* [100].

2. *Install the software.* The user needs to install the software. This may involve installing additional software due to dependencies.

3. *Run the software.* The user needs to start the software at multiple nodes. He can use tools like *pssh* [100].

4. *Monitor the software running.* The user needs to monitor the running to make sure the software works as expected. He may need to debug the code if something is wrong.

It is time-consuming and error-prone for a user to run a large-scale network experiment, as this requires significant amount of work not only in writing the code, but also in setting up the experiment on the testbed nodes and monitoring if it runs as expected. Several specific problems facing the users are listed below. First, it is very common that when a user uploads his program to a number of nodes, some of them will not get the program correctly. There are many possible causes, such as node failures, temporary network problems, etc.

Second, a user often wants to set up his experiment on a number of nodes with certain properties. For example, a DHT designer will want to deploy the experiment to nodes that demonstrate churns. Third, it is not easy to monitor the status of a distributed system as many nodes are involved. All these lead to a need for experiment management facilities.

Several research projects address individual management problems, but none provides a general solution. We believe that the concept of the knowledge plane help us solve the problems as a whole. We conduct a case study on PlanetLab. In the case study, we propose to construct a spec-KP on experiment management that manages distributed experiments on the testbed using knowledge collected and maintained in the spec-KP. The goal is to speed up experiment setup and help users find a node set with desirable features. To reach this goal, we study how different kinds of knowledge are maintained and propagated.

The chapter is organized as follows. Section 6.2 discusses the knowledge collection and distribution related to distributed experiment management problem. Section 6.3 evaluates how well the spec-KP can help reduce the upload failure and find a node set with desirable features. We briefly discuss the related work in Section 6.4. Section 6.5 concludes the chapter.

6.2 Knowledge Management

6.2.1 Goal

In this section, we use the distributed experiment management as an example to demonstrate how knowledge is collected, maintained, and propagated. The purpose of this case study is to see how the knowledge plane can help distributed experiment in two aspects: (1) to speed up the experiment setup by choosing reliable nodes; (2) to automatically choose nodes with desirable properties. To do so, different agents play different roles, and work together to resolve the requests. We focus on knowledge collection and maintenance in this section, and request propagation in the next section.

Property	Value
Desired number of nodes	$\leq$ total available number
Desired node distribution	0-9
Desired node dynamics	0-9

Table 6.1: Request fields.

6.2.2 User Specifications

When a user wants to run a distributed experiment on a testbed, he specifies the desired properties of the node set. Currently two features are supported.

1. The distribution of the nodes. Value between 0 and 9 are used to represent the degree of distribution, where 0 means to choose nodes close to each other while 9 means widely distributed.

2. The dynamics of the nodes. This refers to the reliability of nodes. Similar to the above, 0 means to choose most stable nodes while 9 means most dynamic. This is useful when a user wants to test the system performance under static or dynamic environment.

Accordingly, Table 6.1 demonstrates the fields of a request.

6.2.3 Knowledge Collection

The Spec-KP agents run on every testbed node. They monitor the condition of local hosts, and maintain history records, such as boot time, previous failure causes, etc. Each region collects aggregated information of the local agents. When an experiment is being set up, the spec-KP finds the appropriate nodes based on the user's requests. During this process, different kinds of knowledge are needed, and they are maintained differently. There are two kinds of knowledge we collect for experiment management, according to their locations.

1. Local knowledge about individual hosts and agents. This includes average workload, bandwidth usage, boot history, etc.

2. Network knowledge, including the path condition between two agents, network condition within a region, etc.

The first kind of information is much easier to collect, while the second often requires collaboration among agents. Furthermore, while it is easy for each agent to maintain its local information, it is not obvious how the second kind of information is stored and maintained. For the purpose of demonstration, we choose the first one as a starting place because it is clear.

6.2.4 Global Map Maintenance

To pick a set of nodes, we first need to know the total number of nodes. To do so, we can either collect the node set at runtime or maintain a global node map. We choose the second way as it is faster. The global map is only approximate, representing how many nodes there are in each region. The map is actively maintained among the leaders, but only when the change of the region size reaches a certain percentage (20% in our case study) will the leader broadcast the change to all the other leaders. As the node set of a testbed is relatively stable, and does not change very often, such information has low maintenance cost in our setting.

The global map is an example of the knowledge that does not change frequently, and is widely and frequently used in the experiment setup by all users. Therefore, we choose to maintain it actively at all regional leaders.

6.2.5 Dynamic Knowledge Maintenance

Unlike the global map, other kinds of knowledge are more dynamic, such as the last reboot time, average CPU load, network condition, etc. To pick nodes with desirable features, we need to maintain such history information as well. However, it is not scalable to maintain such information in a centralized way. Instead, we propose that each local node maintain detailed information about itself, and the regional leader maintain aggregated information about the nodes in the region. Leaders do not exchange information actively; instead, they answer requests about it from other agents or leaders.

As there are many different kinds of dynamic node status and different users need different features, it is not efficient to actively maintain such knowledge at leaders. Instead, the leaders only maintain historical data about each node's status.

6.3 PlanetLab Experiments

In this section, we evaluate the effectiveness of the spec-KP in terms of setting up experiment on PlanetLab [96] quickly and finding nodes with desired features.

6.3.1 Experiment Setup

There are about 800 nodes in total on PlanetLab, distributed in 200 ASes. The experiments run on 200 randomly picked nodes. As one of the experiments needs AS path length information, we use RouteViews BGP tables [99] and AS path inference algorithms [56] to find out all the AS paths.

6.3.2 Upload Success Rate

In a large scale network experiment, it often happens that programs are not uploaded correctly, either due to node failure or temporary network problem [65]. Therefore, we prefer a set of reliable nodes to run the experiment. To do so, we leverage the underlying network knowledge plane in two aspects. First, as described in Section 6.2.4, leaders maintain an approximate global map. Given this global map, we assign different numbers of nodes to each region tentatively. The number of nodes assigned to a region is proportional to the region's size. Second, each regional leader tries to find the most reliable nodes in its region using dynamic information. To do so, the leader can either query all members to get the latest information, or use the history data. We choose the latter to reduce the overhead. The procedure is described below and shown in Figure 6-1.

1. A user issues an experiment setup request to the local agent, specifying a number of nodes.

2. The local agent forwards the request to the regional leader.

3. The regional leader analyzes the request, assigns different numbers of nodes to each region using the global map, and sends a request to leaders in each region.

4. Each regional leader picks the most reliable agents in its region using historical data, and returns the list of agents to the originating leader.

5. The originating leader receives a list of nodes from each region, and returns the complete list to the local agent.

6. The local agent forwards the list to the user.

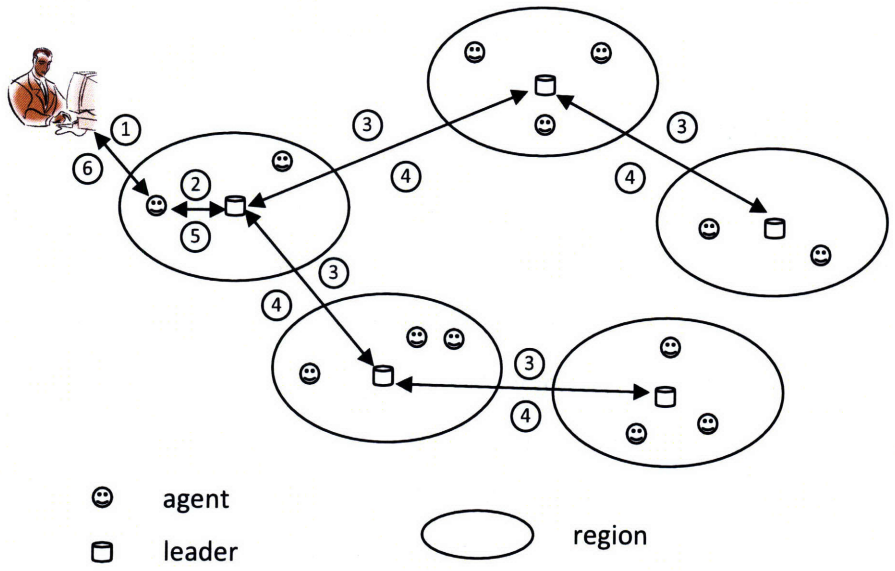


Figure 6-1: Upload procedure.

Figure 6-2 demonstrates the success rate of program uploading in different experiment sizes. We compare our method with random pick. The total number of nodes is 200, and the experiment size starts from 10, increasing by 10 each time. We can see that the random pick has a very stable upload success rate, which is about 70%, no matter what the size is. Our method has a decreasing upload success rate, because we have fewer choices when the size increases. Eventually the two methods meet at 200 nodes.

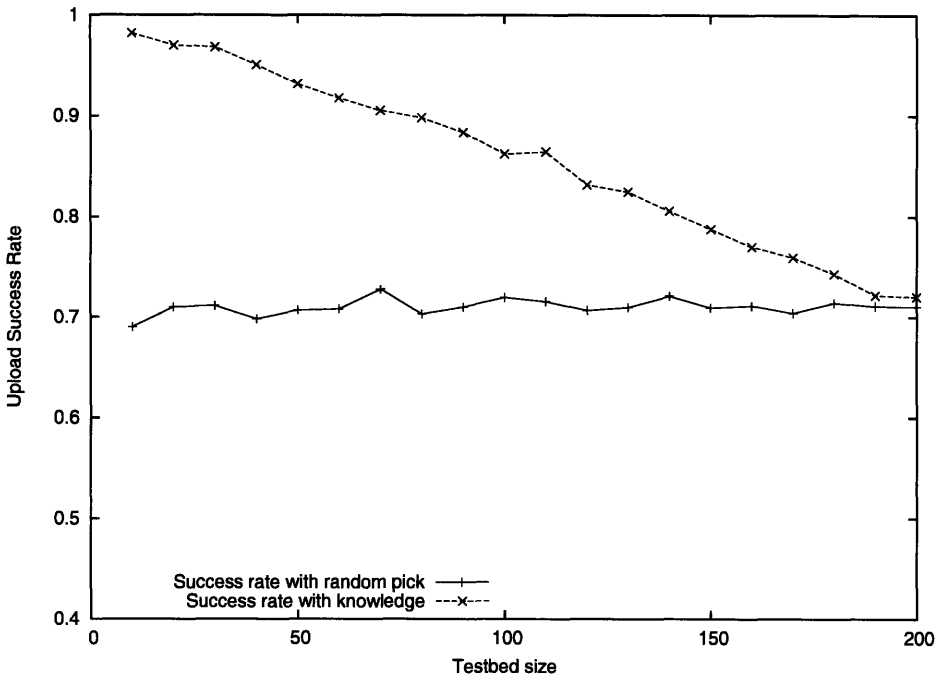


Figure 6-2: Upload success rate measurement.

6.3.3 Node Distribution

The second issue during experiment deployment is that very often we want nodes to be widely distributed in the Internet. To do so, we take advantage of the Autonomous System information. As regions are organized based on both AS and latency information, we can take advantage the static AS topology. We define the degree of node distribution as the average number of AS hops between two nodes.

We compare three node-picking methods. The first is random picking, in which nodes are randomly picked from the global map. The second is "evenly in ASes", in which we try to assign the same number of nodes in each AS, as long as there are enough number nodes there. The third is "greedy", in which we always pick the next node that can maximize the average AS hops. These procedures are similar to the previous experiment, except for two steps. First, when the local leader assigns different number of nodes to each region, it uses the Autonomous System information provided by our AS service. The service provides two functions: the IP to AS number mapping and the AS path between two ASes. We use RouteViews BGP tables [99] and AS path inference algorithms [56] to implement these two functions. In the second method, the leader always tries to pick nodes from dif-

ferent Autonomous Systems, but does not consider the distance between them; in the third method, the leader uses a greedy strategy, which has higher computational overhead, and since the global map is approximate, the leader cannot be too greedy by choosing all nodes from a region, which could lead to the problem of insufficient number of nodes during the actual assignment. Second, each regional leader also tries to find nodes widely distributed in its own region. Figure 6-3 shows how well nodes are distributed using our method compared with random pick. The shape of this figure is similar to Figure 6-2, but the difference between the two methods is less than 1 AS hop. This is because PlanetLab itself is already widely distributed. It has about 800 nodes, distributed in 200 ASes, so nodes by the random pick are likely to be widely distributed. We can improve the distribution if the leader considers the distance between ASes, not just ASes themselves.

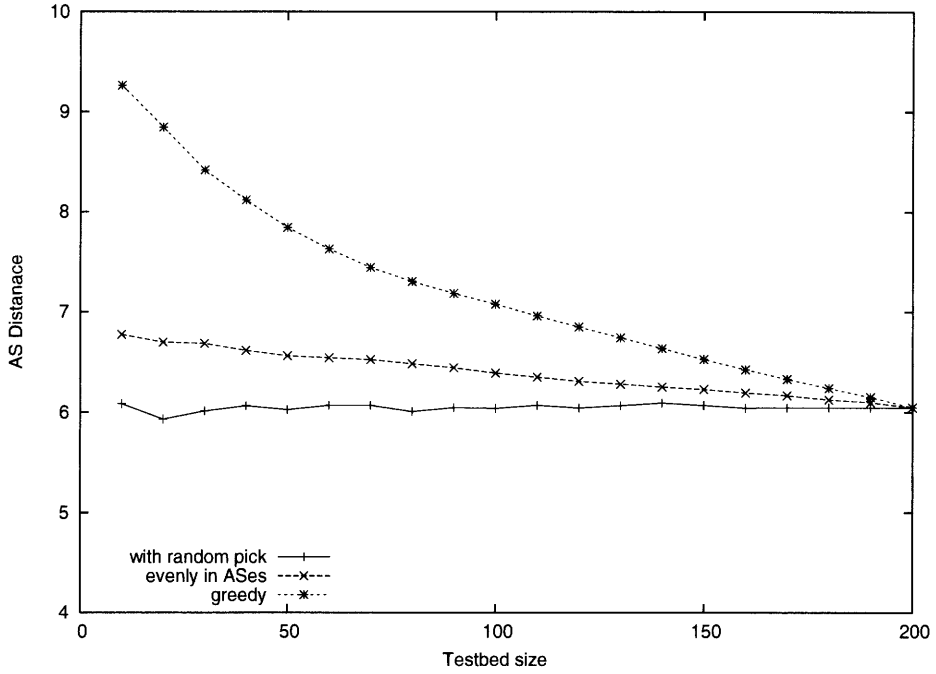


Figure 6-3: Node distribution experiment.

6.3.4 Further Improvements

Besides the above functionalities, we plan to add other useful mechanisms. Within each experiment, agents form a closely coupled region, and share more information with each other. In testbeds like DETER [42] and Emulab [1], the NS script specifies the network

topology. We implement similar functionality on PlanetLab by allowing a user to specify not only the overall properties of the node set to run the experiment, but also the different roles for each node, as very often nodes have different importance in the experiment. Agents monitor the experiment and check if the experiment setup matches the specifications. If something suspicious is found, agents will notify the user. This process can be made interactive so that users may provide more information and agents perform further diagnosis.

To avoid node failure and accelerate the experiment setup process, when an experiment is ready to swap in, agents can collaborate with each other to pick the nodes with the best history records, so that the expected swap-in time will be minimized. During the actual swap-in process, if an agent notices that a host cannot boot correctly, it will pick another available node to replace the problematic one. The agent may also notify the testbed operator about the problematic nodes.

6.4 Related Work

Some recent research projects focus on providing a framework for application management. Plush is a configurable application management infrastructure designed to meet the requirements of distributed applications and execution environments [13]. Plush allows developers to define the flow of control needed using application building blocks. Through a resource management interface, Plush supports execution in a variety of environments, including both live deployment platforms and emulated clusters. Smartfrog consists of a collection of tools that manage distributed applications and a specification language to describe the applications. Applications must follow the API to take advantage of its features [50]. Compared with them, our work is more general in terms of the functionality and requires less integration between our system and the applications. We focus more on the efficient organization of the components and distributed of knowledge among them.

Another category of related work is the efforts to make Emulab-like environment easier to use [113, 43]. The Emulab Experimentation Workbench is designed for replayable network experiments [43]. SEER focuses on security related experimentations, especially

those on the DETER testbed [113]. Both provide convenient and comprehensive facilities to free users from tedious experiment configuration so that they can focus on the key research issue, which is similar in the spirit of this work. But unlike PlanetLab, nodes are located at the same place in the Emulab and the DETER testbed, and as a result, the problems I addressed in this work, such as improving uploading success rate and finding nodes with desired features, either do not exist or require different solutions in those environments.

Some research focuses on failure diagnosis. In [65], researchers analyze the error reporting mechanism in Emulab, and propose a new design on structured error reporting that associates context with each error-type and propagates both error-type and context. They each solve individual problems, but did not address the experiment management problem as a whole from the network architecture point of view. Our work leverages the network knowledge plane to facilitate experiment setup, to reduce upload failures, and to provide new functionalities that are not possible without the knowledge collection and maintenance.

6.5 Summary

In this work we conduct a case study on experiment setup on PlanetLab. Several issues are involved, including different agent roles, region formation and maintenance, knowledge collection and maintenance, etc. More work is needed in the consistency maintenance under network partitioning. The case study demonstrates the effectiveness of the spec-KP for experiment management. Further improvements include support for experiment monitoring and debugging, which needs some specification language to describe the experiments in detail, and thus requires a tighter integration between the spec-KP and the experiment itself.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 7

Case Study II: An Intrusion Detection Framework

The NetKP has provided us a powerful infrastructure to build the spec-KPs for various network management and applications. The premise of the work in this chapter is that it is both feasible and valuable to design a common framework for intrusion detection on top of the NetKP to address several simultaneous and often inter-related problems. We identify three key issues: distributed and broadly valuable knowledge, collected and produced through analysis or diagnosis; a need to respect privacy, security, and proprietary boundaries in the process of intrusion detection; a multiplicity of independent discovery and diagnosis capabilities. In this chapter, we lay out an integrated framework for cooperative and extensible intrusion detection systems. It is our expectation that this framework will facilitate evolution in intrusion detection capabilities, support local privacy and security policies, and scale to the size of the Internet. To demonstrate the strengths of this framework, we build a distributed intrusion detection system on the DETER testbed, and apply different detection techniques within and among regions.

7.1 Introduction

Increasingly frequently and cleverly, worms and viruses are invading our hosts on the Internet. To address this, we see more and more creative intrusion detection systems ap-

plying distinctive techniques, using different data structures for their underlying data, and reporting different kinds of intrusion status [94, 108, 25, 63]. Some are based on traffic patterns, others are based on signatures; some are centralized, and others are distributed. Each of those has its own strengths and weaknesses. Despite the fact that attacks often affect multiple domains or organizations simultaneously, the tools used locally may not support effective collaboration across those boundaries. To date we do not have a way to integrate those techniques and create a comprehensive intrusion detection approach, so that intrusion detection systems widely distributed in the Internet can share information easily and worms be detected quickly. In this work, we propose a knowledge-based framework to unify those systems and make intrusion detection more effective in the Internet.

We classify agents in this framework into detection engines and knowledge agents. Detection engines correspond to the regional leaders, and knowledge agents correspond to the agent members, but in the context of intrusion detection. In order to meet the needs of individual clients and allow for a unified approach, the framework posits that a user contacts one or more detection engines, based on a set of criteria including goals, levels of trust, etc. Each detection engine, whose task is to detect intrusion, in turn will call on a set of knowledge agents with various kinds of expertise, ranging from particular detection techniques to traffic monitoring capabilities. Note that knowledge here includes not only specific intrusion detection techniques, but also information about network configuration, local context, etc. We see as central challenges to this framework the ability to share and discover existing and new knowledge, the choice of an entity to only expose as much or as little of both its interests and its expertise as it chooses, and a means of organizing and cooperating efficiently. For this we identify two key technical support issues: a mechanism for trustworthy private knowledge retrieval, and a scalable organization for agent discovery and knowledge aggregation, as we discuss below.

In terms of security, we face three intertwined problems. First, an agent or source of knowledge may, for policy reasons, want to provide only limited or partial access to the complete set of its possible capabilities. Second, the source of a query may want to disguise its own particular interests from the source of knowledge and expertise. Third, the parties want to have reasonable trust in the veracity of their exchange. In order to address the first

two, we base our work on prior work on Private Information Retrieval, and for trust, on a protocol for developing a trust model.

In terms of scalability, we propose a *divide-and-conquer* approach to global scale organization. We see a number of distinctive motivations all leading to our region-based approach, including simple reduction in scale, often leading to exponential decrease in complexity, and the efficiencies of operating in a homogeneous environment, leading to performance gained by physical or topological partitioning, etc. These sorts of motivations lead us to propose an underlying region-based capability to achieve any or all of these.

This chapter is organized as follows. Section 7.2 describes the framework for composite intrusion detection, then addresses the security issues including protocols of privacy protection and trust, and discusses the region-based organization for agent discovery and knowledge aggregation. Section 7.3 presents a case study on distributed intrusion detection on the DETER testbed based on the framework. Section 7.4 discusses a selection of related work. Section 7.5 concludes the chapter and highlights the future work.

7.2 Knowledge-based Intrusion Detection

7.2.1 Intrusion Detection Framework

Overview

The main purpose of our work is to design a general framework so that existing as well as new knowledge can be integrated into it effectively. The key idea is to treat the result of any detection method or other information (for instance, result of local computations or some prior knowledge like the worm's core code) as a piece of knowledge to be input to the detection engine. As a general term, knowledge in this framework refers to any useful information to intrusion detection in the Internet, including that about individual objects in the network, and the relationships among objects, etc. Figure 7-1 demonstrates this basic framework.

There are three parties in this framework: users, detection engines, and knowledge agents. A user issues an intrusion detection request to a detection engine. The detection

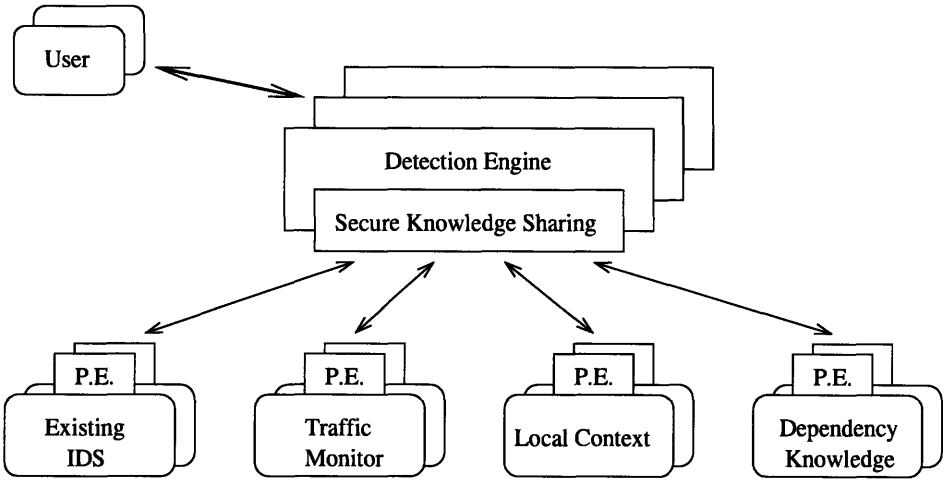


Figure 7-1: Knowledge-based intrusion detection framework. P.E. refers to "policy enforcer" described in Section 7.2.2.

engine analyzes the request, and collects necessary knowledge from knowledge agents and other detection engines. The knowledge agents provide processed knowledge, such as network traffic, local observations, according to their local policies. After collecting enough knowledge, each detection engine builds a dependency graph of the collected knowledge, and then runs inference algorithms on it and reports the result(s) to the user, as demonstrated in an example at the end of this section. All the parties use the same ontology language to describe their requests and capabilities, similar to [69]. We describe each component in detail below. The secure knowledge sharing between parties (dealing with privacy and trust) is discussed in Section 7.2.2, so here we will just mention it at the relevant places.

User

A user can issue a request to a detection engine directly or through a local knowledge agent. A request consists of three parts: *goal*, *constraints*, and *prior knowledge*. The goal defines the task to accomplish, such as determining whether a network is intruded by a worm, etc. The constraints define the conditions that detection engines must satisfy. Prior knowledge provides some existing knowledge that may be useful to the detection engines.

Detection Engine

The task of a detection engine is to detect intrusions. Its role is similar to the regional leader in the previous discussion in the sense that it coordinates the operations of the knowledge agents to resolve a request. A detection engine uses all available knowledge from knowledge agents to resolve a request in the following way. When a request is received, the detection engine analyzes it and figures out which agents are likely to meet the goal and constraints. During this process, the detection engine needs to coordinate the knowledge sharing among agents. Some agents may have relevant techniques, some may have the data, and others may have the dependency knowledge between the techniques. Those with the relevant techniques may not have access to the data due to privacy constraints. The detection engine needs to find proper agents that can run the technique over the data. Then it collects the responses from them, makes a final analysis, returns the result to the user, and caches it for future use. Detection engines are similar to the regional leader discussed in previous chapters, as they coordinate the operations between agents within a network, and communicate with other detection engines to exchange information, but unlike regional leaders, there may be multiple detection engines within a region, as discussed later.

We focus here on the analysis of the detection engine. Once the detection engine has collected some knowledge, it can start to build a dependency graph between the knowledge that it has received so far. During this process, new knowledge from additional knowledge agents may be added, and the detection engine dynamically adjusts the dependency graph. With the addition of new knowledge, there are three possible actions that can result. First, this addition might produce knowledge that the current dependency graph (in this detection engine) does not know anything about. In this case, the most natural way to add this into the current graph is to treat it as new and independent information. Second, this addition might give knowledge relating several pieces of knowledge that are already in the current graph. In this case, the probabilistic structure of the graph will be changed to reflect this new knowledge of the dependency. Third, this addition might give knowledge that does not really change the internal structure of the graph, but instead wraps around it and thus affects the results of the detection engine.

Knowledge Agent

Knowledge agents correspond to the agent members in the previous discussion. Knowledge agents provide various kinds of knowledge, and act as a proxy of users. A knowledge agent can be an existing intrusion detection system that uses specific techniques on an end host or on a network, one that contributes a new detection technique, or one that simply provides any valuable knowledge. We identify four kinds of useful knowledge, as demonstrated in Figure 7-1.

The first and most important category is existing intrusion detection systems. Here we view a detection system as a database of results generated by applying a detection technique on an end host or on the part of the network it has access to. It includes detection techniques based on incoming and outgoing traffic such as [63, 70], and signature-based approaches [25] together with the worm signature databases.

The second is traffic monitor. Such agents monitor the traffic data for the analysis. They often reside on vantage points in the network, such as gateways. Note that in our framework the detection techniques and the data are separate.

The third is local context. This includes network configuration, operating system types, running services, results of local virus scans, etc. Such knowledge exposes the potential vulnerability of the network and hosts.

The fourth is dependency knowledge. This describes the dependency between multiple pieces of knowledge, such as the conditional probability of incoming and outgoing traffic.

When a knowledge agent receives a query from a detection engine, it first checks its local policies on exchanging knowledge with this detection engine. Then it engages in a secure knowledge exchange with the detection engine to provide the knowledge without disclosing sensitive information.

Example

We use a simple example to demonstrate how the components interact with each other in the framework. The example request is to detect whether Code Red intruded the network 1.2.3.4/24 in the past seven days, under the scope constraint that the knowledge agents to

ask must be within the local ISP. A piece of prior knowledge is that the operating system running on most hosts within the network is Windows. The request is shown below. Note that this is just an example to demonstrate how a detection engine may deal with a request, and we do not focus on how a request should be precisely defined in this spec-KP.

```
<Request id=11239045>
     <Goal>
         <Worm> Code Red </Worm>
         <Network> 1.2.3.4/24 </Network>
         <TimeRange> 7 days </TimeRange>
     </Goal>
     <Constraint>
         <Scope> Local ISP </Scope>
     </Constraint>
     <PriorKnowledge>
         <OSType> Windows </OSType>
     </PriorKnowledge>
</Request>
```

Figure 7-2 demonstrates the process. The request is sent by a user to a detection engine. The detection engine parses the request and does the following. We stress again that whenever there is a knowledge exchange, the parties involved use the secure knowledge sharing protocol as described in Section 7.2.2.

1. As the request is about a specific worm, the detection engine checks whether any knowledge agent knows the signature or some properties of Code Red. If not, it has no way to resolve the request, and will return a failure to the user, together with the reason. If the request does not specify any worm, then this step is skipped.

2. If the signature and some traffic pattern are available, the detection engine collects such knowledge, and chooses a number of knowledge agents based on trust, privacy, and the scope constraint specified in the request, through the agent discovery mechanism in Section 7.2.3. Suppose that at this point in time, two agents happen to be chosen, one using signature-based technique, and the other using a traffic pattern based technique. Then the detection engine hands over the knowledge about the worm to the knowledge agents, respectively.

3. The two trustful agents analyze some hosts and the recent traffic in the network using their own techniques, respectively, and return the results. Note that the data analyzed may come from a third traffic-monitor agent.

4. After receiving the results from the knowledge agents, the detection engine builds a dependency graph of the results, and runs some inference algorithm, for instance, Bayesian inference.

   As a concrete example, suppose the detection engine employs the following rule to integrate the results from the knowledge agents: the detection engine will report to the user the probability that both results are "No intrusion". Since there is no dependency knowledge about the results (yet), the detection engine will assume independence between them. Therefore, the final result will be calculated as:

$$\begin{aligned}\mathbf{P}(\text{Intrusion}) &= 1 - \mathbf{P}(\text{R1=No \& R2=No}) \\ &= 1 - \mathbf{P}(\text{R1=No}) \cdot \mathbf{P}(\text{R2=No})\end{aligned}$$

5. At this point in time, another relevant agent happens to join the system, giving dependency knowledge relating the signature and the traffic pattern. For instance, this might be knowledge about $\mathbf{P}(\text{R2=No}|\text{R1=No})$. In this case the result will be revised as:

$$\begin{aligned}\mathbf{P}(\text{Intrusion}) &= 1 - \mathbf{P}(\text{R1=No \& R2=No}) \\ &= 1 - \mathbf{P}(\text{R1=No}) \cdot \mathbf{P}(\text{R2=No} \mid \text{R1=No})\end{aligned}$$

6. The detection engine reports a final result to the user.

This example demonstrates how a detection engine detects an intrusion with the collaboration of multiple knowledge agents, while following the privacy and other constraints, and how a new piece of knowledge helps the detection engine obtain a better result. Note that a technique itself is just a piece of knowledge in this framework, and this is especially useful when a detection engine could not let the agent with that technique analyze the data directly due to privacy constraints.

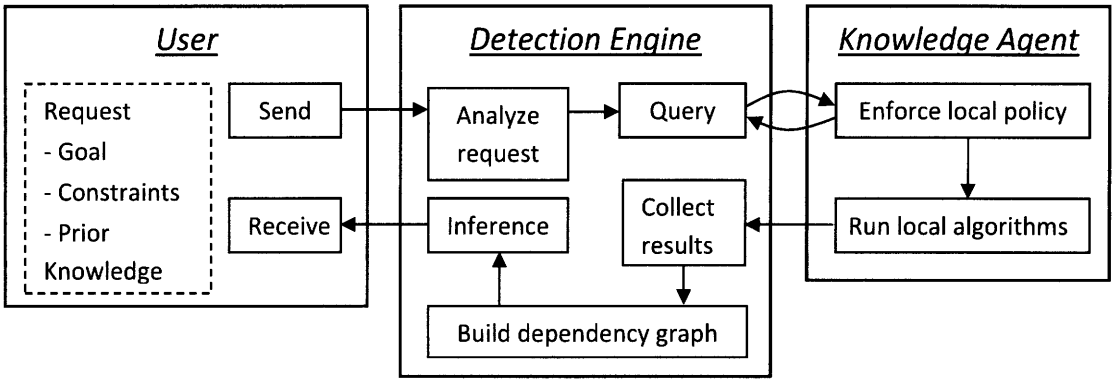


Figure 7-2: The resolution process of a request.

7.2.2 Secure Knowledge Sharing

Different parties may be under different administrations, and would not disclose sensitive information to others. Furthermore, agents have different capabilities and credentials. Therefore, we need a mechanism for secure knowledge sharing that protects sensitive information for both sides (knowledge provider or receiver) and help establish a trust system to represent capabilities and prevent cheating. The mechanism discussed in this section is an important part in the knowledge-based framework, and the following discussion presents the basic idea of private information retrieval and how it is applied in this framework, but more work is needed to make it practical, and this mechanism is not implemented in the intrusion detection system in Section 7.3.

Private Knowledge Retrieval

Allman et al. proposed a loose private matching scheme for knowledge sharing in [15], which allows for information sharing about mutually observed network events. Here we generalize and extend it using Private Information Retrieval (PIR) and policy enforcers. The goal is as follows. First, PIR enables detection engines to encode a query in such a way that knowledge agents can answer the query but do not know the query itself. Second, policy enforcers on the knowledge agents make sure only proper knowledge will be sent to the detection engines. Therefore, little sensitive information is revealed to both sides.

Private Information Retrieval (PIR) has been extensively studied in theoretical com-

puter science [32, 68]. For this work we choose a computationally bounded PIR approach because the alternative requires either complete copies of the database at both ends or transmission of the complete database, both infeasible in this work.

Although PIR can protect privacy of both the detection engines and the knowledge agents, agents often have their own policies about what knowledge can be exposed to which detection engines. Therefore, each agent needs a policy enforcer (P.E. in Figure 7-1). The policy enforcer implements the security policy to prevent the exposure of sensitive local information, but to allow for the report of valuable non-sensitive knowledge to the detection engines. For instance, an enterprise might not allow any information about local detection, but might be prepared to allow for a report that it has authoritatively identified an attack with a particular signature. In return for allowing that information out, it can expect to receive similar information from other enterprises or business entities, without having to expose anything about the nature of its network or about the degree of compromise it might have experienced.

In the face of knowledge sharing between mutually distrustful parties, each detection engine needs to send queries of some form to knowledge agents (which can be seen as databases in this respect). However, we do not wish for the knowledge agents to know which particular entries the detection engine is interested in, lest it provide false information. To achieve this, we can apply the ideas of PIR.

We sketch the ideas of PIRs and then move on to describing the application of PIRs to our setting. Where possible we refrain from providing formal definitions and theorems, but instead describe, at a high level, particular (theoretical) implementations.

As background, a number y is called a quadratic residue modulo n (QR) if there exists an integer x such that $y \equiv x^2 \ (mod\ n)$. Otherwise, y is called a quadratic non-residue (QNR). It is a number theoretic fact that the product of any number of QRs is still a QR, and the product of any odd number of QNRs is a QNR. Let n be a composite of the form $p \cdot q$, where p and q are primes of the same size. A particular implementation of PIR [68] relies on the following cryptographic assumption. *Quadratic Residuosity Assumption*: without knowing p and q, no polynomial time Turing machine can distinguish a random QR (modulo n) from a random QNR (modulo n) while with p, q, it is easy to distinguish them.

Assuming each entry in the database is a single bit, 0 or 1. The detection engine wants to know the bit at the ith entry in the database. The detection engine sends to the database a query vector which contains a distinct QR for each index other than i and a QNR for the ith element. The database cannot distinguish which are the QRs and which the QNR. All it does is to compute the product of all those vector elements for which the corresponding database entry was a 1. The detection engine then can test this product for whether it is a QR or a QNR and hence know whether the ith entry was a 0 or a 1, but the database has no clue which entry is actually queried.

To make the scheme more practical, we need to use an extension to the PIR called *private keyword search* [31, 46], which allows querying the database if a keyword exists in it. A particular implementation is to translate private keyword search into the above PIR scheme, as follows. Assuming the database holds t possible keywords in total, and they are of the same length l bits, the database first sends all the t keywords to the detection engine. The detection engine replaces the t keywords with their incidence vector, a 2^l bit string in which the jth bit is 1 iff the jth l-bit string, in the lexicographic order, is one of the existing keywords. Suppose the keyword the detection engine holds, W, is the kth word in the lexicographic order. Then it becomes a problem of retrieving the kth bit of the 2^l bit string from the database, the same as the basic PIR. The scheme can be made more efficient with an *oblivious walk*, which is beyond our discussion. Please refer to [31, 46] for detail.

More work is needed to make PIR practical so that it can be applied to regular queries. In the following we will use PIR as a black box and demonstrate how to use it in our protocol below. Say that a detection engine DE wishes to query the knowledge agents $KA_1, KA_2, ..., KA_m$. The protocol is as follows. For concreteness we focus on the particular example in which database entries are of the form "(IP, Port, Protocol, Traffic, Time)":

Protocol KnowledgeSharing

1. *User Input.* The user provides the detection engine DE with the goal, constraints, and prior knowledge. The detection engine DE then contacts each knowledge agent $KA_i, i \in 1, ..., m$ that it thinks it would need knowledge from, to inform them to start the knowledge handshake.

2. *Knowledge Handshake.* For each $i \in 1, ..., m$, KA_i checks its local policy regarding information exchange with DE, then KA_i computes some function f_i on its database D_i based on the policy, to end up with $f_i(D_i)$. Then KA_i tells DE the form of queries allowed.

3. *Knowledge Query.* DE sends each KA_i a query that it is interested in, as a function of the user's inputs, conforming to the form that KA_i deemed as valid. Here the detection engine can use the PIR mechanism to protect its privacy, if needed.

4. *Knowledge Answer.* KA_i computes and sends the results using the PIR, and DE extracts the answer using the PIR.

Note that in step *Knowledge Handshake*, if KA_i has no local policy that restricts information exchange with DE, then f_i is the identity function, and KA_i would tell DE that the valid queries are of the form "(IP, Port, Protocol, Traffic, Time)"; if there is a local policy that only allows aggregate information exchange with DE without disclosing the IP address, port number, or protocol type, then KA_i would tell DE that the valid queries are of the form "(TotalTraffic, StartTime, EndTime)".

Dealing with trust

We have addressed privacy issues by assuming each party tells the truth. However, in reality our framework consists of many parties with different interests, and adversaries may intentionally join the framework and provide false information. In this section, we demonstrate how to integrate trust into our framework, independently of any underlying trust model. Note that in the literature, usually *trust* is a subjective viewpoint of one's capability, defined between two parties, while *reputation* is an objective measure for one's capability, defined globally [7]. For our purposes we do not (need to) distinguish trust and reputation. Below we describe how trust is used in our framework:

1. A user chooses a number of detection engines based on their rankings of trust.

2. Based on the user's rating, the detection engine may accept or reject the request. If a request is accepted, the detection engine chooses a number of knowledge agents to

ask based on their rankings.

3. Knowledge agents accept or reject the requests from the detection engine based on the rating.

4. The user rates the detection engines based on the quality of the returned results using some out-of-band method and sends feedback to the detection engines.

5. Based on the user's feedback, each detection engine rates the knowledge agents involved in this process, and differentiates them based on the quality of the knowledge they provided. It also forwards the rating information to the knowledge agents.

6. The knowledge agents rate the detection engine based on the received rating information and other available information such as the request rate.

7. The rating process can be interactive by designing an interactive protocol for two parties to argue about the feedback, either between the user and the detection engines or between a detection engine and the knowledge agents. The parties periodically exchange their ratings.

Note that our framework is independent of the trust model we use, except possibly the feedback mechanism – if the trust model has a feedback mechanism, it could be used in the protocol described above; otherwise, a feedback mechanism has to be designed for the particular trust model.

For concreteness, we propose a simple trust model as follows. To start, a user or a detection engine may trust certain knowledge agents more, such as those officially deployed by his institution or ISP. Then the rating by user U of detection engine E at time $t+1$ will be:

$$
\begin{aligned}
{}_{t+1}R_E^U = w_1 \cdot \Big(w_2 \cdot {}_tR_E^U + (1-w_2) \cdot \sum_i {}_tR_{U_i}^U * {}_tR_E^{U_i} \Big) \\
+ (1-w_1) \cdot {}_{t+1}f_E^U
\end{aligned}
$$

where U is the current user, the U_i are the other users that the current user U knows, E is the detection engine U is interested in, w_1 is the weight to incorporate the feedback,

$0 \leq w_1 \leq 1$, w_2 is the weight to incorporate previous rankings, $0 \leq w_2 \leq 1$, ${}_tR_Y^X$ is the trust that party X has on party Y at time t, f_E^U is the feedback from E at time $t+1$. The above formula shows that the current rating of a detection engine is a weighted sum of its previous rating from the users and the current feedback. Ratings of other entities including knowledge agents are calculated in a similar way.

7.2.3 Scaling and Organization

In order for a user to contact useful detection engines, for a detection engine to find expert agents, and for the knowledge agents to discover and utilize knowledge, we propose a rendezvous approach that involves agent discovery and knowledge aggregation. To address issues of scaling, efficiency, and a variety of policy constraints, we again use the *region* to be a first-class component in the framework. Below we touch first on our region approach and second on our rendezvous for discovery and aggregation technique.

Region and constraints

For scalability and efficiency, we follow a divide-and-conquer strategy by dividing the framework into regions. As discussed in previous chapters, the region is a new design element in the network architecture that encapsulates and implements scoping, subdividing, and crossing boundaries of sets of entities [117]. Detection engines and knowledge agents are organized into regions based on four kinds of constraints:

1. *Functional constraints.* Intrusion detection is conceived as a set of interacting components collocated with the knowledge necessary to succeed. Although in our framework detection engines and knowledge agents are distributed, there are constraints from the knowledge itself as well as the functional subcomponents and their interactions.

2. *"Network" location.* In some cases, intrusion detection may by necessity be kept local to that network, where locality may be defined by a number of metrics, such as network topology, latency, etc.

3. *Physical location.* We separate this from the previous category because the issues of geographic location or perhaps administrative ownership boundaries are generally orthogonal to the more performance based network location constraints.

4. *Policy and other external constraints.* Security constraints are a significant factor in organizing entities. Furthermore, in order to meet real world requirements, an organizational strategy must be able to integrate economic, regulatory and other constraints.

We propose a simple clustering scheme as a starting point of the regionalization approach. Based on its own constraints, each entity in the framework decides which other entities to connect to. Entities connected to each other form a region, and select a leader for management purpose. Connections among regions are maintained by the leaders.

Discovery and aggregation

We face two complementary problems: agent discovery and request/knowledge aggregation. There has been a significant amount of work in the areas of publish/subscribe systems and peer-to-peer systems towards addressing these problems, but scaling and heterogeneity problems remain. In this effort we intend to build those mechanisms in the context of our regionalization approach.

In this framework, requests are issued by users directly or through local knowledge agents. Requests are sent to the detection engines. The detection engine analyzes the request and sends further requests to the knowledge agents based on their advertised capabilities. We propose an agent discovery scheme similar to that in the NetKP: Each detection engine maintains a directory service of knowledge agent types and capabilities within the local region. All the detection engines form a tree structure similar to that in Chapter 5. When a detection engine cannot find needed knowledge within the local region, it can propagate a request in the tree. The request can be either propagated to the whole tree, or to detection engines within n hops in the tree, where n is determined by several factors such as priority, cost, etc.

Aggregation is another important function, as many similar requests may be issued when a fast-propagating worm attacks. We take advantage of the region structure and the cross-region organization in the aggregation. Assume that the detection engines form a tree structure globally. A simple version of aggregation is a multi-layer aggregation: First, within a region, if a detection engine receives multiple similar requests from the local knowledge agents, it does not propagate all of them, but aggregates those requests and forwards only one request. Second, at the region level, if a detection engine receives similar requests from other detection engines, it can hold the later requests and waits for the first one to be answered, and returns the answer to all the requesters.

7.3 A Dependency-based Intrusion Detection System

In this section, we design and implement a collaborative intrusion detection system based on the framework proposed above. Not all the components in the framework are implemented in this system, and we customize the framework to fit our needs. The goal in this system is to detect zero-day, slow-scanning worms, for which no existing signatures are available. We organize end hosts into regions based on existing partitions in the Internet, which we posit is positively correlated to the existing dependency structure. Detection engines and knowledge agents run on end hosts. Leveraging on this organization, we apply different intrusion detection techniques within and across regions. We use a hidden Markov model (HMM) within a region to capture the dependency among hosts, and use sequential hypothesis testing (SHT) globally to take advantage of the independence between regions. We conduct experiments on the DETER testbed, and preliminary results show improvement on detection effectiveness and reduction of communication overhead.

Traditionally, intrusion detection is carried out at a central point, usually a gateway, as it is a natural position to observe incoming and outgoing traffic. This approach is prone to DoS attacks, and depends on non-local detection of anomalies, prompting a need for new approaches to monitor and respond to security incidents. To that end, host-based distributed intrusion detection has been a promising direction. A key challenge in such a distributed intrusion detection system is that end hosts need to be organized efficiently and intrusion

detection techniques applied effectively, so that an intrusion detection decision can be made before the worm infects most of the hosts. Many current mechanisms use simple gossiping protocols or peer-to-peer protocols [76, 40, 30] to aggregate local determinations.

Since the behaviors of zero-day worms are not known a priori, the best location for initial attention is the local host itself, in the context of local behavior and applications [40, 30]. However, at the local node, one loses the aggregation effect of repeated or simultaneous low level anomalies. In addition, it is difficult to make local detectors strong because they see only a small percentage of the global traffic. Thus, to detect intrusions effectively, we must aggregate the results of weak local detectors to get a broader perspective. This work addresses the question of algorithms for effective aggregation. Improving local detection is a separate problem that we do not address here.

New intrusion detection techniques are needed to deal with different dependency structures among hosts more effectively, and the work in Section 7.2 provides a suitable framework to do so. We postulate an observable causal relationship between the success likelihood of a particular intrusion attempt and network proximity between end hosts. This is based on the observation that enterprise networks are reflected in topological neighborhoods, and also likely to be supporting many similarly configured and managed machines, thus repeating the same weaknesses across an enterprise. Thus, if one host in an enterprise is susceptible in a certain way, it is more likely that its peers are as well. In contrast, random hosts far away from each other in the broad Internet are likely to be independent of each other. Therefore, we can take advantage of different dependency structures between hosts with different detection techniques. In addition, worms often scan consecutive IP addresses, which causes another kind of dependency. For example, Code Red II chose a random IP address from within the class B address space of the infected machine with probability $\frac{3}{8}$; with probability $\frac{1}{2}$ it chose randomly from its own class A; with probability $\frac{1}{8}$ it would choose a random address from the whole Internet [119].

We believe that a good distributed intrusion detection system should satisfy two key requirements: (1) efficient host organization based on network proximity and dependency, and (2) detection techniques that leverage this host organization and the dependency structure. In this work, we propose a dependency-based host organization and message propa-

gation protocol. End hosts are organized into cooperating regions based on their network proximity and policy boundaries. Then different detection techniques are applied at different levels. Each end host runs a weak local detection system. For a region, we use a discrete-time Hidden Markov Model (HMM) with unsupervised learning to estimate intrusion status of that region (this captures the dependency) [101]. At the global level, we use a sequential hypothesis testing (SHT) globally to coordinate findings across regions (this takes advantage of the independence) [64]. We implement our mechanism on the DETER testbed [42], and evaluate the performance of this system and the communication overhead. Experiment results show that our mechanism can detect intrusion faster, better and cheaper.

In this work we only evaluate time-homogeneous first order HMMs (where the transition probabilities between the different states do not vary with time), and use a simple static organization based on both dependency and network proximity. Non-homogeneous higher order HMMs, based on an adaptive organization utilizing various kinds of network knowledge, will be considered in future work.

7.3.1 Host Organization

To build an effective intrusion detection system, we propose an agent organization based on the concept of regions, and discuss the communication mechanism among different types of agents.

Regions

We organize agents into a two-level hierarchy based on the enterprise network boundaries, for the following reasons. First, as mentioned before, enterprise networks are likely to support many similarly configured and managed machines, thus repeating the same weaknesses across an enterprise. Second, due to the security and policy constraints, hosts within an enterprise network are allowed to share detailed information, while hosts in different enterprises usually cannot. Third, enterprise networks are reflected in topological neighborhoods, so hosts in the same enterprise are usually close to each other, and thus the communication among them is more efficient.

Second, the leaders organize themselves into a communication structure. If the number of leaders is small, the leaders form a complete graph; otherwise other organizations the tree structure proposed in Chapter 5. Figure 7-3 demonstrates a region-based organization. It consists of three regions. Agents close to each other are clustered into the same region.

Corresponding to the region-based structure, we classify the agents into three kinds of detectors: local detectors, regional detectors, and global detectors. A local detector resides on each host. Regional detectors are similar to the regional leaders. The global detectors may reside on any hosts. There may be one or more global detectors, depending on the requirement on robustness and the communication structure.

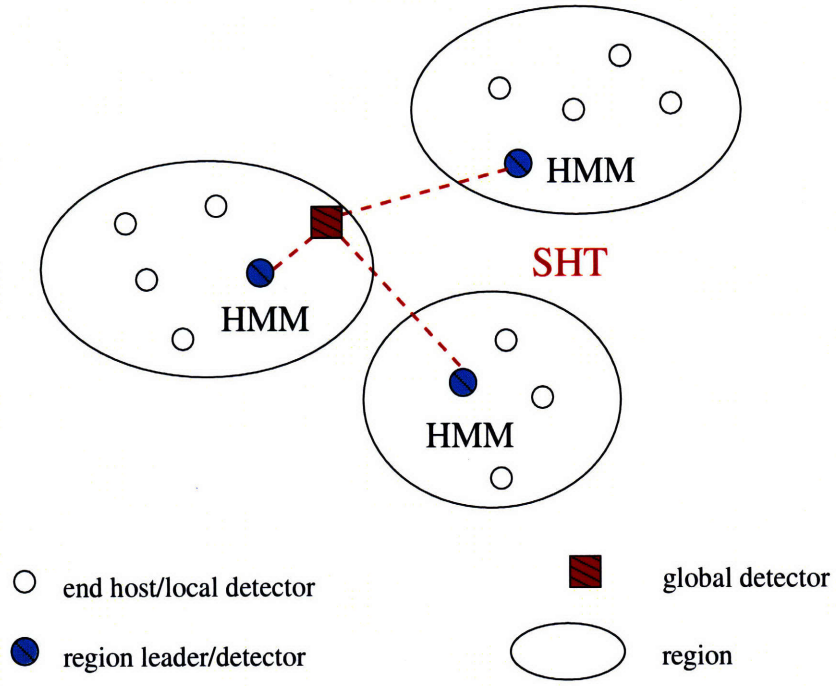


Figure 7-3: A region-based organization example.

Communication

Local detectors only communicate with their regional detector. When a local detector detects a potential intrusion attempt, it sends an alert to its regional detector directly. The regional detector collects alerts from local hosts, runs its regional detection algorithm, and then reports to the global detectors. Global detectors wait for reports from multiple regional detectors, and run the global detection algorithm.

Depending on the tradeoff between robustness and overhead, there may be different communication structures between regional detectors and global detectors. For example, we can deploy only one global detector, and all the regional detectors report to it. This centralized method has low communication overhead, but the global detector may become the target of DoS attacks. As another extreme example, we can have one global detector on each regional leader, together with the regional detector. Each regional detector multicasts its report to all the global detectors. Therefore, each region (through its global detector) has a global view of the intrusion situation. Whenever a global detector has enough information to make a decision, it announces its decision to the other global detectors and all the regional detectors. We could also have chosen an intermediate position in which there was more than one global detector, but not as many as one per region.

7.3.2 Intrusion Detectors

As mentioned above, there are three kinds of detectors in our system: local detectors, regional detectors, and global detectors. Each kind of detector runs the appropriate algorithm, as described in this section.

Local detector

A local detector is an instantiation of the knowledge agent in the framework in Section 7.2. It resides on each end host. These are weak in their capability of detecting intrusions, and as stated earlier the design of local detectors is a separate problem that we do not address here. The detection criteria may vary, depending on each host. For concreteness, we use the following simple local detector in our experiments: when an end host receives a packet at an un-serviced port, the corresponding local detector triggers an alert to its regional detector; otherwise, it sends a clean signal. There are two things to note. First, there are both false positives and false negatives in the signals the local detector sends. Second, there is a tradeoff between timeliness and detection overhead. If the end host sends one signal upon receiving every packet, the overhead may be too high. If the end host batches signals, this causes a delay in the detection. As alerts are more important than clean signals, we can

send out alerts immediately, but batch clean signals.

Regional detector

A regional detector is an instantiation of the detection engine that collects knowledge within its region. It diagnoses potential intrusions at the neighborhood level, using discrete-time Hidden Markov Models (HMMs) to detect intrusion for each region. We choose to use HMMs instead of SHTs, because, as discussed above, we believe that the probability of effective intrusion between close neighbors can be dependent on that proximity, and HMMs allow us to reflect that. The second advantage of the HMM approach is the ability to capture a notion of time and therefore multiple connection attempts to the same host. In contrast, SHT systems are particularly easy to game: the worm can make sure that the first connection attempt to any host is always to a servicing port. This is because SHT systems can only handle the first connection attempt to any host, lest the independence assumption breaks down.

Figure 7-4 demonstrates an HMM for a region. It has four states: *00, 01, 10, 11*, representing a value pair of *(infected?, suspicious?)*. The first bit represents whether there is a worm in the region, and the second bit represents whether there is some host whose behavior is suspicious. This captures the adaptivity of the worm in the sense that an infected host can decide to lay dormant for the time being to avoid detection (similarly, a clean host might accidentally behave suspiciously). Higher order models can be used to capture more of the adaptivity. In general, the model parameters are unknown and have to be estimated. Each regional detector uses the reports (alert or clean) from local hosts to Baum-Welch train the model and to generate the Viterbi path of hidden states [101]. This Viterbi path gives the most likely sequence of hidden states that could have generated the observed sequence of triggering of the local detectors, under the current estimated parameters. Note that the HMM models the current incoming traffic pattern, so it does not matter whether the region is under one worm attack or simultaneous worm attacks.

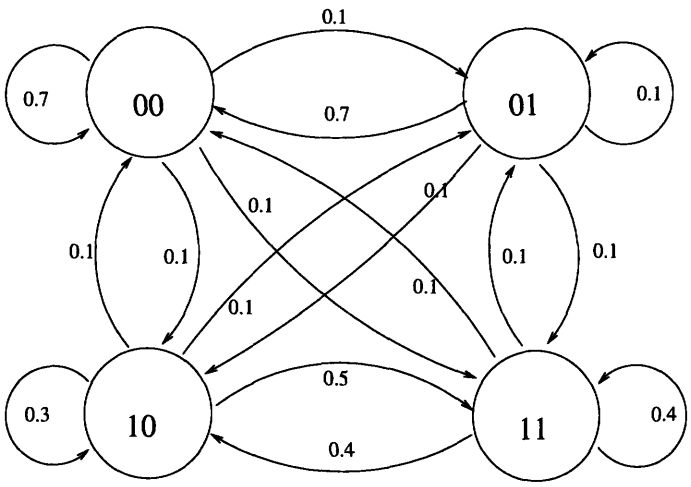


Figure 7-4: Initial Hidden Markov Model at regional detectors. *00* means the region is clean and its behavior is not suspicious, *01* means clean but suspicious, *10* means infected but not suspicious, *11* means infected and suspicious.

Global detector

The global detector is also an instantiation of the detection engine, except that it collects knowledge globally and employs a different detection technique. The global detector uses sequential hypothesis testing (SHT) to determine whether there is an intrusion at the global level, because we believe that under a good organization, different regions can be assumed to be independent of each other in terms of intrusion conditions. Therefore, we use SHT with the independence assumption, and always use the newest information from each region as input to the SHT. The following equation of $L(\bar{Y})$ defines the likelihood ratio from the observation vector $\bar{Y} = \{Y_1, Y_2, ..., Y_n\}$, given two hypotheses H_0 ("no intrusion") and H_1 ("intrusion"), respectively. Y_i indicates whether the regional detector at region i believes there is an intrusion (1) or not (0). Note that $P[Y_i = 0|H_1]$ is the probability of false negative, and $P[Y_i = 1|H_0]$ is that of false positive.

$$\begin{aligned} L(\bar{Y}) &= \frac{P[\bar{Y}|H_1]}{P[\bar{Y}|H_0]} \\ &= \frac{P[Y_1|H_1] \cdot P[Y_2|H_1] \cdots P[Y_n|H_1]}{P[Y_1|H_0] \cdot P[Y_2|H_0] \cdots P[Y_n|H_0]} \end{aligned}$$

Then $L(\bar{Y})$ is compared with the lower and upper thresholds. The thresholds, T_0 and T_1, are calculated by two parameters: desired detection rate, DD, and desired false alarm rate, DF, as follows:

$$T_0 = \frac{1 - DD}{1 - DF},\ T_1 = \frac{DD}{DF}$$

If $L(\bar{Y})$ is less than T_0, then the global detector accepts hypothesis H_0; if $L(\bar{Y})$ is greater than T_1, then H_1 is accepted; otherwise, i.e., $L(\bar{Y})$ is between T_0 and T_1, no conclusion is made and further observations are taken. For the details of SHT, please refer to [125].

Here we just clarify on one potentially confusing aspect of the above equations. If we have a higher desired detection rate DD, it seems a bit odd that the upper threshold T_1 would increase, because this seems to leave more infections undetected. The intuitive way to explain this is that we also have to look at the lower threshold as well since both of the thresholds matter, and once the SHT reaches a decision the case is closed. The mathematical way to explain this is that the SHT guarantees certain bounds on the true detection rate (TD) and the true false positive rate (TF), given the desired ones, as follows:

$$TF < \frac{1}{T_1}; TD > 1 - T_0.$$

Therefore if DD increases,T_1 increases and the upper bound on TF decreases, whereas T_0 decreases and the lower bound on TD increases. So, as to what really happens when we change our desired rates DD and DF, SHT does not guarantee that increasing DD will increase TD, only that the lower bound on TD increases.

7.3.3 Performance Evaluation

In this section we present our experiments on the DETER testbed. We choose to do our experiments on DETER instead of simulation, as the former is more realistic and may

provide more insights. Our evaluation consists of two parts. The first is the effectiveness of our on-line detection mechanism, in which we evaluate the performance of both regional HMM (rHMM) and global SHT (gSHT). The second is the efficiency of region-based host organization, in which we measure the detection speed and communication overhead.

Experiment Setup

Our experiments run on 88 nodes in the DETER testbed. Nodes are clustered into 8 regions, 11 nodes each, and the links between regions are slower than those within a region. Worms are emulated using WormSim [79], and we implement a special worm that scans sequentially within a region and randomly chooses the next region to scan, thus creating dependency with a region and independence between regions. Background (clean) traffic is generated on each node at a constant rate. There are both false positives and false negatives. That is, normal traffic may be mistaken as intrusion attempts, and intrusion attempts may be viewed as clean. Nodes are divided into two categories: vulnerable and non-vulnerable. Vulnerable nodes will be infected when an intrusion attempt arrives, and then the worm will propagate from the infected host. Non-vulnerable nodes will issue an alert when receiving an intrusion attempt. WormSim and local detectors run on all the nodes except the regional leader nodes. Regional detectors run on the regional leaders, one for each region. rHMM is implemented using the General Hidden Markov Model library (GHMM) [2]. In this experiment, there is only one global detector.

Intrusion Detection Performance

In this experiment, we evaluate the performance of our system. The regional detectors run rHMM, and the global detector runs gSHT over all regions. Table 7.1 lists the parameters used in rHMM and gSHT. As we described in Section 7.3.2, a regional detector trains the model and infers a Viterbi path. Given the Viterbi path, there is still a question of how to determine whether the region is under intrusion or not. In this work, we use a simple empirical algorithm: if the latest six states contain three consecutive intrusion states (*11* or *10*), then there is an intrusion. Recall that *11* means that the rHMM thinks that (some nodes of) this region is infected and suspicious activity is detected, and *10* means that this

region is infected and currently exhibiting normal behavior.

Regional Hidden Markov Model (rHMM)	
Noise level	0.03
Initial transition matrix	see Figure 7-4
Initial state probability	{0.7, 0.1, 0.1, 0.1}
Global Sequential Hypothesis Testing (gSHT)	
False positive	0.10
False negative	0.01
Desire false alarm rate	0.02
Desire detection rate	0.98
Experiment settings	
Number of regions	8
Number of nodes per region	11
Vulnerable nodes	25%
Worm propagation rate	1 scan/second
Normal traffic rate	1 message/second

Table 7.1: The rHMM and gSHT experiment parameters.

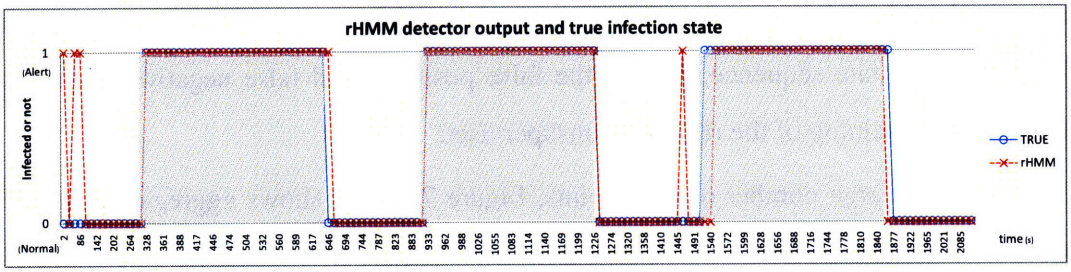


Figure 7-5: Regional Hidden Markov Model performance. *TRUE* is the states based on the true status (no false positive or false negative), and *rHMM* is that based on the observations of a regional detector. Gray areas represent actual infection periods.

Figure 7-5 shows the intrusion detection of a region using rHMM. The experiment is divided into alternate clean periods (blank areas in Figure 7-5) and infection periods (gray areas). *0* means clean and *1* means infected. The solid line (*TRUE* in the figure) shows the true intrusion status (i.e., no false positives or false negatives). The dashed line (*rHMM*) shows the detection result of an rHMM using the reports from local detectors. Note that the time axis is not linear due to data aggregation. We can see that at the beginning of the experiment, rHMM makes a few mistakes due to the false positives and the relatively untrained model. However, it learns to correct the errors very soon. Although there are

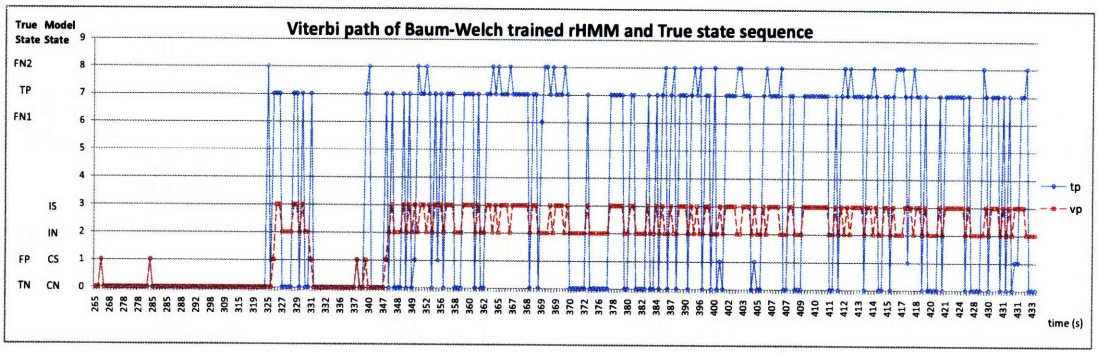


Figure 7-6: Viterbi path of an rHMM and the true state sequence. *tp* is the true state/event sequence, and *vp* is the inferred state sequence by the rHMM. In the dashed line, *0,1,2,3* in Y-axis correspond to the four states in Figure 7-4: clean and not suspicious (CN, corresponds to state 00), clean but suspicious (CS, to state 01), infected but not suspicious (IS, to state 10), and infected but suspicious (IS, to state 11). In the solid line, *0,1,6,7,8* represents an event that a host receives the following packets respectively: a normal packet (true negative, TN), an alert caused by false positives (FP), a clean signal caused by false negatives (false negative case 1, FN1), a true alert caused by an intrusion attempt (true positive, TP), a clean signal from a vulnerable host who cannot/would not distinguish intrusion attempts from normal traffic (false negative case 2, FN2).

some noticeable lags, overall, rHMM's performance is very close to the model trained using the true state sequence, despite the false positives and false negatives. We stress again that the training of the rHMM is unsupervised.

Due to the large number of data points, Figure 7-5 only shows aggregate results. To look at how well an rHMM works in detail, we compare a sequence of true states with the predicted sequence of states from an rHMM in Figure 7-6, which is between 265 and 433 seconds in Figure 7-5. Local detectors report an alert (*1*) or a clean signal (*0*) to the regional detector, which may be a false positive or a false negative. Corresponding to that, the solid line shows the true state sequence. Its states labeled *0,1,6,7,8* represents that a host receives the following packets respectively: a clean signal, an alert caused by false positives, a clean signal caused by false negatives, a true alert, a false negative from a vulnerable host who cannot tell intrusion. The dashed line shows the transition of states observed by the rHMM. The states {0, 1, 2, 3} correspond to the four states {00, 01, 10, 11} in Figure 7-4. We can see that at the beginning there are two small spikes, and rHMM considers it clean but suspicious. When intrusion really happens at 325 seconds, the true sequence jumps to state 8 and then 7; the inferred Viterbi path first jumps to 1, thinking that

it might be just a clean host that accidentally acted suspicious. As more alerts are received, it realizes that the region is under attack, and the state oscillates between states 2 and 3. This in particular means that when the rHMM thinks the region is under attack, but normal packets are received, the rHMM thinks that the worm is laying dormant, as opposed to the region being clean. After that, it remains between 2 and 3 during the infection period, and is not affected by the false negatives and normal traffic. Figure 7-7 demonstrates the new rHMM model after the experiment.

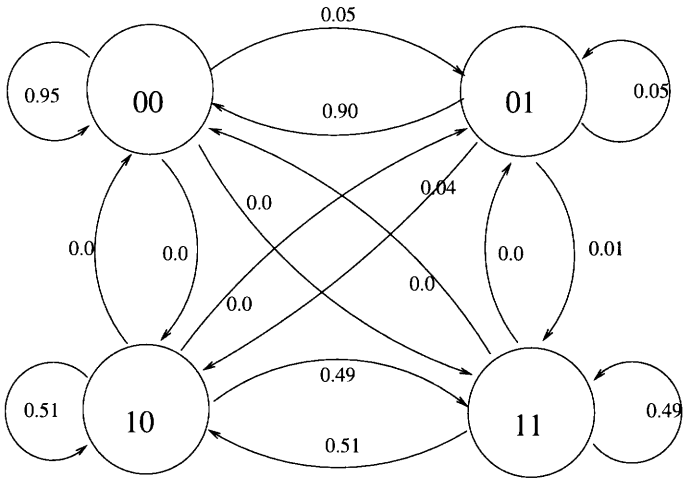


Figure 7-7: A trained hidden Markov model at a regional detector.

Figure 7-8 demonstrates the detection performance of the global detector using Sequential Hypothesis Testing (gSHT): the solid line is the result using the true states and the dashed line is that using rHMM outputs. Since they are quite close, this shows that as far as the gSHT is concerned, the rHMM outputs are almost as good as the truth, lagging a little bit behind.

Compared with Figure 7-5, gSHT does not have the false positives at the beginning and near 1445 seconds in rHMM. This is because that the global detector collects information from multiple regions, and the independence of regions helps eliminate the false positives.

Region-based host organization

To evaluate the efficiency of region-based host organization, we compare our method with a gossiping protocol in three aspects: detection speed, communication overhead, and cost.

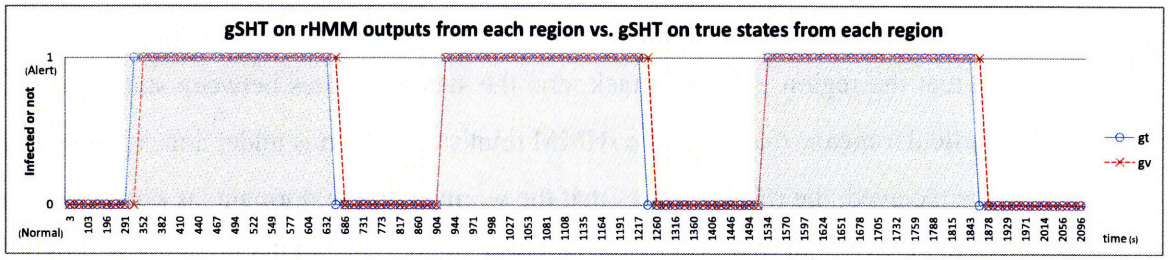


Figure 7-8: Global sequential hypothesis testing performance. *gt* is gSHT's decision based on rHMM outputs, and *gv* is gSHT's decision based on the true states from each region. Gray areas represent actual infection periods.

Detection speed measures how fast hosts make a decision on intrusion detection. Communication overhead refers to the number of messages that hosts propagate to reach a decision. Cost is the number of nodes infected by the time of detection.

One set of the experiment results is shown below in Figure 7-9. *Gossip* refers to the gossiping protocol in [30], where hosts/local detectors exchange observations using an epidemic spread protocol without any organizing structure, and the local detectors run sequential hypothesis testing using the received results. The gossiping rate is 2, which means a local detector forwards the results it computes or receives from others to 2 randomly picked local detectors. *Region* refers to our region-based protocol. We can see that *Region* outperforms *Gossip* in all the three metrics. *Region* is faster in detection time, because alerts are aggregated within each region first before being processed at the global detector, while in *Gossip* messages may cross slow links many times between hosts before reaching a decision. Similarly, the number of infected nodes is also smaller in *Region* than in *Gossip*. Finally, the number of messages transmitted in *Region* is significantly smaller than that in *Gossip*. The reason for this is because that the number of messages increases almost exponentially among hosts in *Gossip*, while in *Region* messages from end hosts are only sent to the regional detector, and then processed at the global detector after aggregation.

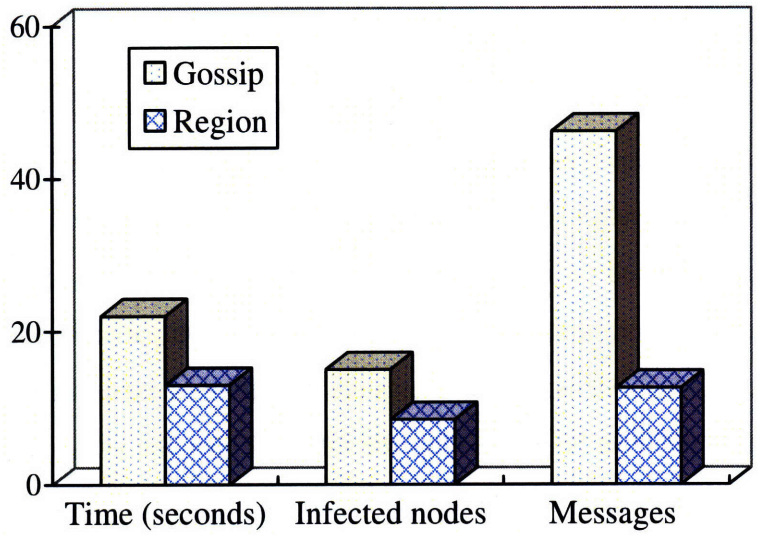


Figure 7-9: Detection speed and overhead comparison. Note that the messages do not include those for maintaining the cooperation among hosts in their method *Gossip* or messages for training the rHMMs in our method. The number of messages is in units of 10.

7.3.4 Discussions

Robustness and Flexibility

Our method is semi-centralized, and can be made more robust to DoS attacks in two ways. First, instead of having only one global detector, multiple widely distributed regional detectors can exchange information so that everyone has an approximate global view and make the decision, thus reducing the vulnerability. Second, regional detectors can be periodically re-selected distributedly, so it is hard for attackers to predict the leaders when attacks happen.

There are two kinds of dependency to be recognized. One is the dependency between end hosts, caused by their proximity, similarity of hardware, software, management, and policy boundaries, etc. Therefore, we assume that network proximity is positively correlated to the dependency structure. The other is the dependency caused by worm scan: for instance, worms may scan an IP block each time, or intentionally scan hosts distant from each other. To deal with this, our approach provides for the flexibility to re-organize regions by considering both kinds of dependency.

DETER testbed

Our experience with the DETER testbed shows that DETER provides a valuable infrastructure for security-related experiments. We suggest several possible improvements here. First, it would be very helpful if DETER incorporated more security-related facilities, such as traffic generator based on real traces, worm simulators, etc. This would greatly simplify the design of experiments and provide the basis for comparison of results among researchers. Second, NS extension commands are important to experiment automation. We hope more commands can be provided in the future. Third, the swap-in process can take a long time when experiments scale up. A way to automatically kill the preloaded experimental programs and reload everything without swap-in or rebooting would significantly reduce the waiting time and speed up the experiment process.

Further Improvements

Our experiments on the DETER testbed suggest that dependency-based host organization can improve intrusion detection by providing valuable network-layer and application-layer knowledge to intrusion detection systems. In the future we will follow up with a series of further experiments:

1. Use HMMs across different regions, to confirm that across different regions, there is essentially no loss of effectiveness if we assume independence. Worms with different scanning features will be tested.

2. Use more general HMMs, specifically a non-homogeneous higher order HMM, based on an adaptive organization utilizing different network knowledge. If a strong global clock is available, then continuous time HMMs can be used too.

3. Enhance the reporting scheme to record the signature of a worm. This will provide two improved capabilities. The first is to significantly reduce the impact of false positives. The second is to improve the reporting of a worm, by allowing for reporting of a particular worm signature, thus enabling the disentanglement of simultaneous worm intrusions.

7.4 Related Work

7.4.1 Intrusion Detection Framework

Lee designed a common architecture for distributed probabilistic Internet fault diagnosis in [69], where new knowledge can be easily added to the diagnosis process. We adopt a similar approach, but unlike fault diagnosis on reachability in Lee's work, intrusion detection techniques are more diverse and heterogeneous. In our system, a priori we have no clue of what the knowledge would be, and dependency relationships can be changed based on new knowledge. For example, two previously-assumed independent pieces of knowledge may be discovered to be dependent given new information; or the system might get a piece of knowledge that it has not seen before.

Several architectures and mechanisms have been proposed to detect intrusion [114, 14, 15, 128]. In [15] Allman et al. proposed a distributed architecture with cross-organizational information sharing to fight coordinated attackers. Their system consists of "detectives" and "witnesses". The detectives are savvy network monitors equipped with sophisticated intrusion detection techniques, while witnesses that are widely distributed in the Internet provide simple observations to detectives. Information sharing between detectives and witnesses is through loose private matching. Our work improves on three aspects. First, our framework is more general. Information is generalized and unified as knowledge, be it the result of heavy-weight detection engines, the result of simple local detectors, or even prior information about the network or the attack. We believe that most knowledge comes from the edges: the existing IDSs. In contrast, the witnesses are relatively simple in their architecture. Second, information sharing is done by secure knowledge sharing. Third, we consider (re)organizations of the entities in our framework to improve the efficiency of agent discovery and knowledge aggregation.

7.4.2 Intrusion Detection Techniques

Our main focus is on zero-day, slow-scanning worms, as in [119, 30, 11]. Such worms propagate themselves slowly to avoid attention caused by dramatic traffic increase. There

are no signatures available as they are completely new.

Many intrusion detection techniques have been developed. Anything based on prior knowledge, such as signature-based approaches [94, 108, 25], cannot be used against zero-day worms since there is no prior knowledge available in a zero-day intrusion.

Bayesian network based techniques are used in [40] to imbue end hosts with probabilistic graphical models. With random messaging to gossip state among the local detectors, they show that such a system is able to boost the weak local detectors to detect slowly propagating worms.

Sequential hypothesis testing (SHT) was first adopted to intrusion detection by Jung et al. in [64]. The original algorithm was centralized, with detection performed at the gateway. It was decentralized in [30], where hosts exchange their information, and perform the inference individually in parallel. We identify two issues with this approach. First, it assumes independence among intrusion attempts and, second, it cannot deal with the case when a worm interleaves the intrusion traffic with non-intrusion traffic. In our work, we assume dependence among hosts within a region, and assume independence between regions. To address this dependence/independence, we use a Hidden Markov Model (HMM) to detect intrusion within a region and SHT globally among regions. The HMM allows us to incorporate our dependency assumption into the regional aggregations, and SHT depends on our assumption of independence between regions.

Machine learning has been applied to intrusion detection in various aspects. For example, Agosta et al. designed an adaptive mechanism that adjusts the threshold of anomaly based on traffic [11]. This does not seem to handle alternating traffic either. Our use of the HMM approach allows us to handle such interleaving, because it learns both transition and emission probabilities from observations, since neither is known a priori.

7.4.3 Communication Protocols

In a centralized intrusion detection system such as [64], all the information is collected and processed at a central point. In a collaborative intrusion detection system, end hosts need to communicate with each other to pool their information together.

Various communication protocols have been applied to distributed intrusion detection systems. One is centralized where all local detectors report intrusion information to a global detector. A recent innovation is to use gossiping protocols among local detectors or multiple global detectors [30, 40].

In [30], decision making is completely distributed. Hosts exchange observations using an epidemic spread protocol without any organizing structure. When a potential intrusion is detected by an end host, it forwards an alert to m randomly selected neighbors, and then each neighbor forwards the alert to its m neighbors together with its own observations, and so on. Each host computes the possibility of intrusion using all the information it has received. This continues unless a decision is made by a host. Usually m equals 1 or 2 for scalability reasons. Each host computes the possibility of intrusion using all the alerts it has received plus its own conclusion. If a host believes that there is an intrusion, it will broadcast its decision to all hosts. In contrast, a set of global detectors are used in [40] with a gossiping protocol.

To the best of our knowledge, previous systems have not considered host organization to achieve more effective detection and efficient communication. Therefore, the communication can be inefficient. More importantly, intrusion detection techniques often assume independence in the intrusion attempts amongst all hosts. This is unlikely to be true when nearby hosts are scanned by a worm. In our method, we make use of the network topology and dependency information to organize a region, and consider the dependency among hosts within each region. In this sense, our method can be seen as a hybrid between a centralized and a distributed intrusion detection system.

7.5 Summary

The strength of this framework comes from its generality and extensibility to support intrusion detection using a wide range of detection techniques and knowledge in a secure and efficient way. Achieving this goal requires overcoming several challenges, and we address three key issues: knowledge-based framework, secure knowledge sharing and scalable organization.

We developed a prototype to demonstrate the strength of our framework using existing techniques on the DETER testbed. First, we design a dependency-based host organization for collaborative intrusion detection. Hosts are clustered into regions based on network proximity and dependency, and communication among them becomes more efficient.

Second, we apply different intrusion detection techniques within regions and across regions. At the regional level, due to the proximity and dependency between hosts, we use a Hidden Markov Model. At the global level, due to the distance and independence between regions, we use sequential hypothesis testing.

The experiments conducted on the DETER testbed show that our mechanism can not only improve the effectiveness of the intrusion detection, but also speed up the detection process and reduce the communication overhead.

Our experimental results suggest that dependency-based host organization can improve intrusion detection by providing valuable network-layer and application-layer knowledge to intrusion detection systems.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The ultimate goal of the knowledge plane is to build a new generation of network that can drive its own deployment and configuration, that can diagnose its own problems, and make decisions to resolve them. There are many challenging issues, such as knowledge representation and utilization, trust and security, economic incentives, etc. As a step towards the knowledge plane, in this thesis I propose a network knowledge plane at the network layer, address several key issues in it, and conduct case studies on the spec-KPs. According to the end-to-end arguments [111], only common and essential functions should be put into the network layer, while in this research I propose to add more functions to the network layer. I believe that as the Internet becomes increasingly pervasive, more and more applications need to learn more about network conditions to work correctly and efficiently besides end-to-end connectivity. Therefore, we need a common infrastructure to provide such network knowledge and mechanisms, at low cost, for applications, and hence it does not contradict the end-to-end arguments.

In this thesis, I first propose an architecture that consists of a network knowledge plane and, on top of it, several specialized KPs. The NetKP provides network knowledge and facilities to help construct the spec-KPs. Each spec-KP is specialized in its own area of interest under application-dependent constraints. I further analyze the organizing criteria and principles to build such an infrastructure, and propose a region-based organization.

Second, I focus two important issues in the NetKP. One is cross-region organization. I design a distributed hash table that leverages network topology knowledge, in which a hybrid proximity neighbor selection algorithm helps achieve scalability, robustness, efficiency and non-intrusiveness. The other is a broadcast and aggregation mechanism among regions. I design a robust tree construction and maintenance protocol using parent functions. The parent function family allows the efficient construction of multiple interior-node-disjoint trees, thus preventing single points of failure commonly in the tree structures.

Third, I conduct two case studies on the spec-KPs: experiment management on testbeds and distributed intrusion detection. In the first case study, I study how to facilitate distributed experiment management on PlanetLab, specifically on how different kinds of knowledge are maintained and propagated to resolve node selection problem during the experiment setup. In the second case study, I design a framework for collaboration between different intrusion detection systems, and implement a distributed intrusion detection system on the DETER testbed. The system organizes agents into regions following corporate network boundaries, applies different detection techniques within and among regions, and takes advantage of the dependency among hosts.

By designing and implementing the NetKP and conducting case studies on spec-KPs, I hope to improve our understanding of the knowledge plane, and to motivate future research in this area.

8.2 Future Work

The core research in this thesis is to define, design, and demonstrate the network knowledge plane and its supporting mechanisms that provide the ability to improve network management and facilitate network application organization. I have studied several important problems in the knowledge plane, but many challenging questions still need further exploration. The following is an incomplete list of questions in the future work:

1. Besides providing knowledge and facilities proposed in this work, what other mechanisms/utilities should be provided by the NetKP to support the spec-KPs? For example, anycast or multicast primitives are useful for the spec-KPs to build their own

specific anycast or multicast mechanisms or the agent discovery mechanism.

2. Spec-KP organization needs to be explored further, especially with respect to request propagation. Can we find a scalable way to create a gradient for request resolution in the Internet like those in directed diffusion [61]? How do we manage all dimensions of knowledge?

3. I believe a standard is needed to describe both facts and cause-effect graphs. This is important for an agent to be able to figure out, given a problem, what facts to collect, where to collect those facts, and how to reason using the facts.

4. We made certain choices about the definition of regions. A future piece of research is to discuss if the choices are the best and what are the alternatives for different scenarios.

5. There are still many issues we have not addressed in the framework for distributed intrusion detection, such as the ontology language, a practical PIR method, etc. Although our framework is proposed for intrusion detection, we believe the design is general enough for many other large-scale network systems that involve different parties and information sharing among them.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix A

Convolutions in the Accuracy Analysis in Chapter 5

As described in Section 5.2.6 in Chapter 5, in order to analyze the accuracy of this approach, we make the following assumptions. We divide time into equal intervals of length C_r, and there will be one refresh/probe in each interval. The time that the refresh/probe happens is uniformly distributed within each interval. Therefore, the probability density function (pdf) is constant at $1/C_r$ in each interval. The lifetime of any node follows an exponential distribution with parameter λ_l, as the exponential random variable is a good model for the amount of time until a piece of equipment breaks down or until an accident occurs. The aggregation events are a Poisson process with rate parameter λ_a, as a broadcast or aggregation request can be considered as an arrival and the Poisson process is natural to describe probabilistic arrival events.

Consider the beginning of a refresh interval and label that time 0. Let R be the time interval from 0 until the next refresh. Let L equal the time interval from 0 until the next failure. Since the lifetime distribution is exponential and thus memoryless, the distribution of L is again exponential with parameter λ_l. Let A equal the time interval from 0 until the next aggregation. Likewise, the inter-aggregation interval is exponentially distributed and thus memoryless, the distribution of A is again exponential with parameter λ_a. For simplicity, we condition on the aggregation occurring at time a, and that all the different characteristics of different nodes are independent, unless stated otherwise.

We consider the conditional probability that the aggregation result is correct from a single node's perspective, i.e. its parent has received its aggregation result.

All the probabilities we talk about below are conditioned by the fact that the aggregation occurred at time a. By the total probability theorem, we can split this into different (sub)cases.

1. *Case 1: Parent does not fail within time 0 to time a.* Conditioned further on this, the probability of aggregating correctly for a node is 1. The probability of this case is $\mathbf{P}[L > A|A = a] = 1 - \int_0^a \lambda_l e^{-\lambda_l l} dl = e^{-\lambda_l a}$.

2. *Case 2: Parent does fail sometime in between time 0 to time a.* The probability that the parent fails sometime in between time 0 to time a is $\mathbf{P}[L \leq A|A = a] = 1 - e^{-\lambda_l a}$. There are two subcases to consider, whether the sequence of events is: failure, refresh, aggregation (*Case 2-1*), or refresh, failure, aggregation (*Case 2-2*). In the latter case the result would be incorrect, so we do not need to calculate it. In the former case, we make another simplification: the refresh always has enough time to complete before the aggregation (so refresh is instantaneous). This can be further divided into two worlds.

   (a) *Case A:* $a \leq C_r$.

   In essence the probability we are dealing with now is

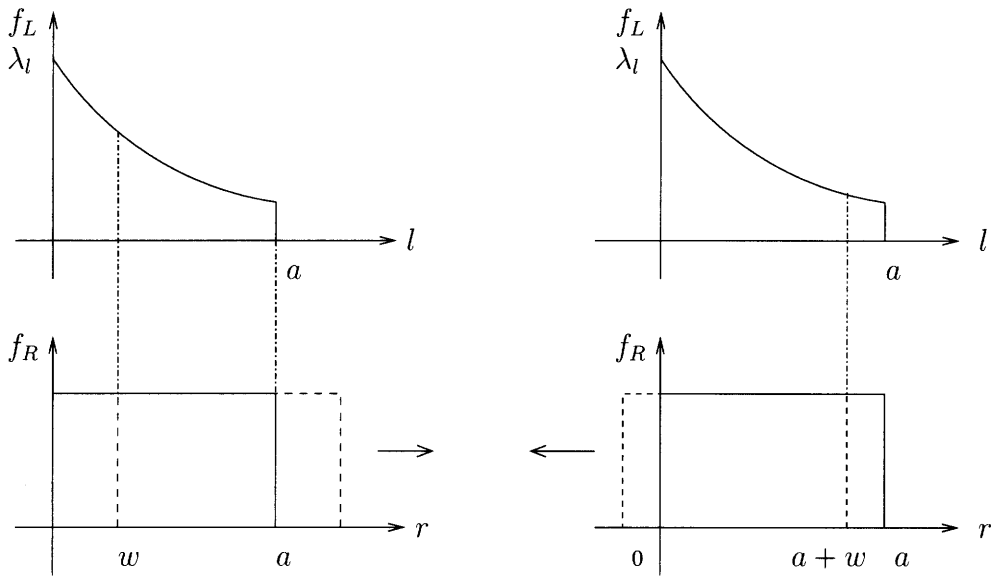
   $\mathbf{P}[\text{Aggregation correct for a node}|A = a, L \leq A, a \leq C_r]$
   $= \mathbf{P}[L \leq R \leq A|A = a, L \leq a, a \leq C_r]$
   $= \mathbf{P}[L \leq R|L, R \leq a \leq C_r] \cdot \mathbf{P}[R \leq A|A = a, L \leq a, a \leq C_r]$
   $= \mathbf{P}[L - R \leq 0|L, R \leq a \leq C_r] \cdot P[R \leq a|a \leq C_r]$
   $= \mathbf{P}[L - R \leq 0|L, R \leq a \leq C_r] \cdot \frac{a}{C_r}$

   (again, L, R denotes the lifetime and the refreshing-time r.v.s respectively). Let W denote the random variable $L - R$. Using graphical calculation of convolutions [22], we identify the following cases:

   i. When $w \geq a$ or $w < -a$, f_W is 0 because either f_L or f_R is 0.

ii. When $0 \leq w < a$, the plot f_R is shifted to the right. Figure A-1a demonstrates the integral range of l in this case.

iii. When $-a \leq w < 0$, the plot f_R is shifted to the left. Figure A-1b demonstrates the integral range of l in this case.



(a) $0 \leq w < a$ (b) $-a \leq w < 0$

Figure A-1: Convolution in case A. The dashed rectangle shows the shift of f_R due to w.

Then we get the following as the pdf of W after doing the convolution $f_W = \int_l f_L(l) f_R(l-w) dl$:

$$f_W = \begin{cases} 0 & \text{if } w \geq a \\ \int_{l=w}^{a} \frac{1}{a} \frac{1}{1-e^{-\lambda_l a}} \lambda_l e^{-\lambda_l l} dl & \text{if } 0 \leq w < a \\ \int_{l=0}^{a+w} \frac{1}{a} \frac{1}{1-e^{-\lambda_l a}} \lambda_l e^{-\lambda_l l} dl & \text{if } -a \leq w < 0 \\ 0 & \text{if } w < -a \end{cases}$$

Note that in the above $f_L(l)$ and $f_R(l-w)$ are under condition $L \leq a$ and $R \leq a$, respectively. This evaluates to

$$f_W = \begin{cases} 0 & \text{if } w \geq a \\ \frac{1}{a}\frac{1}{1-e^{-\lambda_l a}}(e^{-\lambda_l w} - e^{-\lambda_l a}) & \text{if } 0 \leq w < a \\ \frac{1}{a}\frac{1}{1-e^{-\lambda_l a}}(1 - e^{-\lambda_l (a+w)}) & \text{if } -a \leq w < 0 \\ 0 & \text{if } w < -a \end{cases}$$

Therefore,

$\mathbf{P}[L \leq R \leq A | A = a, L \leq A, a \leq C_r]$

$$\begin{aligned} &= \mathbf{P}[L - R \leq 0 | L, R \leq a \leq C_r] \cdot \frac{a}{C_r} \\ &= \frac{a}{C_r} \int_{w=-a}^{0} \frac{1}{a} \frac{1}{1 - e^{-\lambda_l a}} (1 - e^{-\lambda_l (a+w)}) dw \\ &= \frac{1}{C_r} \frac{1}{1 - e^{-\lambda_l a}} (a - \frac{1}{\lambda_l} + \frac{1}{\lambda_l} e^{-\lambda_l a}) \end{aligned}$$

(b) *Case B:* $a > C_r$.

Here, we are after the same probability

$\mathbf{P}[\text{Aggregation correct for a node} | A = a, L \leq A, a > C_r]$

$= \mathbf{P}[L \leq R \leq A | A = a, L \leq A, a > C_r]$

$= \mathbf{P}[L \leq R | L, R \leq a, a > C_r] \cdot \mathbf{P}[R \leq A | A = a, a > C_r, L \leq R]$

$= \mathbf{P}[L - R \leq 0 | L, R \leq a, a > C_r] \cdot P[R \leq a | a > C_r]$

$= \mathbf{P}[L - R \leq 0 | L, R \leq a, a > C_r] \cdot 1$

$= \mathbf{P}[L - R \leq 0 | L, R \leq a, a > C_r]$

Again, we let W denote the random variable $L - R$, and carry out the convolution, and get:

i. When $w \geq a$ or $w < -C_r$, f_W is 0 because either f_L or f_R is 0.

ii. When $a - C_r \leq w < a$, the plot f_R is shifted to the right. Figure A-2a demonstrates the integral range of l in this case.

iii. When $0 \leq w < a - C_r$, the plot f_R is shifted to the right. Figure A-2b demonstrates the integral range of l in this case.

iv. When $-C_r \leq w < 0$, the plot f_R is shifted to the left. Figure A-2c demonstrates the integral range of l in this case.

$$f_W = \begin{cases} 0 & \text{if } w \geq a \\ \int_{l=w}^{a} \frac{1}{C_r} \frac{1}{1-e^{-\lambda_l a}} \lambda_l e^{-\lambda_l l} dl & \text{if } a - C_r \leq w < a \\ \int_{l=w}^{C_r+w} \frac{1}{C_r} \frac{1}{1-e^{-\lambda_l a}} \lambda_l e^{-\lambda_l l} dl & \text{if } 0 \leq w < a - C_r \\ \int_{l=0}^{C_r+w} \frac{1}{C_r} \frac{1}{1-e^{-\lambda_l a}} \lambda_l e^{-\lambda_l l} dl & \text{if } -C_r \leq w < 0 \\ 0 & \text{if } w < -C_r \end{cases}$$

Note that in the above $f_L(l)$ and $f_R(l - w)$ are under condition $L \leq a$ and $R \leq a$, respectively. This evaluates to

$$f_W = \begin{cases} 0 & \text{if } w \geq a \\ \frac{1}{C_r} \frac{1}{1-e^{-\lambda_l a}} [e^{-\lambda_l w} - e^{-\lambda_l a}] & \text{if } a - C_r \leq w < a \\ \frac{1}{C_r} \frac{1}{1-e^{-\lambda_l a}} [e^{-\lambda_l w} - e^{-\lambda_l (C_r+w)}] & \text{if } 0 \leq w < a - C_r \\ \frac{1}{C_r} \frac{1}{1-e^{-\lambda_l a}} [1 - e^{-\lambda_l (C_r+w)}] & \text{if } -C_r \leq w < 0 \\ 0 & \text{if } w \leq -a \end{cases}$$

Therefore,

$\mathbf{P}[L \leq R \leq A | A = a, L \leq A, a > C_r]$

$$\begin{aligned} &= \mathbf{P}[L - R \leq 0 | L, R \leq a, a > C_r] \\ &= \int_{w=-C_r}^{0} \frac{1}{C_r} \frac{1}{1 - e^{-\lambda_l a}} (1 - e^{-\lambda_l (C_r+w)}) dw \\ &= \frac{1}{C_r} \frac{1}{1 - e^{-\lambda_l a}} (C_r - \frac{1}{\lambda_l} + \frac{1}{\lambda_l} e^{-\lambda_l C_r}) \end{aligned}$$

We can now calculate the probability of the aggregation being correct from a single node's perspective, applying independence:

$$\begin{aligned} &\mathbf{P}[\text{Aggregation is correct for a node} | \text{Aggregation occurred at } a] \\ &:= \mathbf{P}[\text{correct} | A = a] \\ &= \mathbf{P}[\text{correct} | A = a, L > A] \cdot \mathbf{P}[L > A | A = a] \\ &\quad + \mathbf{P}[\text{correct} | A = a, L \leq A] \cdot \mathbf{P}[L \leq A | A = a] \\ &= 1 \cdot e^{-\lambda_l a} + \mathbf{P}[\text{correct} | A = a, L \leq A] \cdot (1 - e^{-\lambda_l a}) \end{aligned}$$

At this point, depending on whether we are in case A ($a \leq C_r$) or case B ($a > C_r$), the result is going to be different. Recall that we are fixing $A = a$, so we are either in case A or case B.

In case A ($a \leq C_r$) we have:

$$\begin{aligned}
\mathbf{P}_A &= \mathbf{P}[\text{correct}|A = a] \\
&= 1 \cdot e^{-\lambda_l a} + \frac{1}{C_r}\frac{1}{1 - e^{-\lambda_l a}}(a - \frac{1}{\lambda_l} + \frac{1}{\lambda_l}e^{-\lambda_l a}) \cdot (1 - e^{-\lambda_l a}) \\
&= e^{-\lambda_l a} + \frac{1}{C_r}(a - \frac{1}{\lambda_l} + \frac{1}{\lambda_l}e^{-\lambda_l a}) \\
&= \frac{1}{C_r}(a - \frac{1}{\lambda_l}) + (1 + \frac{1}{C_r\lambda_l})e^{-\lambda_l a}
\end{aligned}$$

and in case B ($a > C_r$) we have:

$$\begin{aligned}
\mathbf{P}_B &= \mathbf{P}[\text{correct}|A = a] \\
&= e^{-\lambda_l a} + \frac{1}{C_r\lambda_l}e^{-\lambda_l C_r} - \frac{1}{C_r\lambda_l} + 1
\end{aligned}$$

Note that the inter-arrival time between the aggregation events is exponentially distributed with parameter λ_a. Therefore, we can combine the two cases and get the final formula. Let Z be the event that the aggregation is correct for a node. Table 5.1 shows the probabilities of several typical settings.

$$\begin{aligned}
\mathbf{P}[Z] &= \mathbf{P}[\text{Aggregation is correct for a node}] \\
&= \int_0^{C_r} \mathbf{P}_A f_A(a)da + \int_{C_r}^{\infty} \mathbf{P}_B f_A(a)da \\
&= \int_0^{C_r} (\frac{1}{C_r}(a - \frac{1}{\lambda_l}) + (1 + \frac{1}{C_r\lambda_l})e^{-\lambda_l a})\lambda_a e^{-\lambda_a a}da \\
&\quad + \int_{C_r}^{\infty} (e^{-\lambda_l a} + 1 - \frac{1}{C_r\lambda_l} + \frac{1}{C_r\lambda_l}e^{-\lambda_l C_r})\lambda_a e^{-\lambda_a a}da \\
&= \frac{1}{C_r\lambda_a} - \frac{1}{C_r(\lambda_a + \lambda_l)} + \frac{\lambda_a}{\lambda_a + \lambda_l} - \frac{1}{C_r\lambda_a}e^{-\lambda_a C_r} + \frac{1}{C_r(\lambda_a + \lambda_l)}e^{-(\lambda_a + \lambda_l)C_r}
\end{aligned}$$

To understand the final result, let us look at several extreme situations:

1. When $\lambda_l \rightarrow 0$, P[Z] $\rightarrow 1$. $\lambda_l \rightarrow 0$ means the node life time goes to infinity. P[Z] approaches 1, because the aggregation will always be correct when there is nearly no node failures.

2. When $\lambda_a \rightarrow \infty$, P[Z] $\rightarrow 1$. $\lambda_a \rightarrow \infty$ means that the inter-aggregation interval goes to 0. In this case, P[Z] approaches 1 because when the aggregation event happens very frequently, the probability that it happens before the node failure approaches 1.

3. When $C_r \rightarrow \infty$, P[Z] = $\frac{\lambda_a}{\lambda_a+\lambda_l}$. When C_r goes to infinity, the probability of aggregating correctly is equal to the probability that the aggregation event happens before the node failure event. As the two events are modeled as independent Poisson process, the merged process is a Poisson process with rate $\lambda_a + \lambda_l$, and the probability that the first arrival is an aggregation event is $\frac{\lambda_a}{\lambda_a+\lambda_l}$.

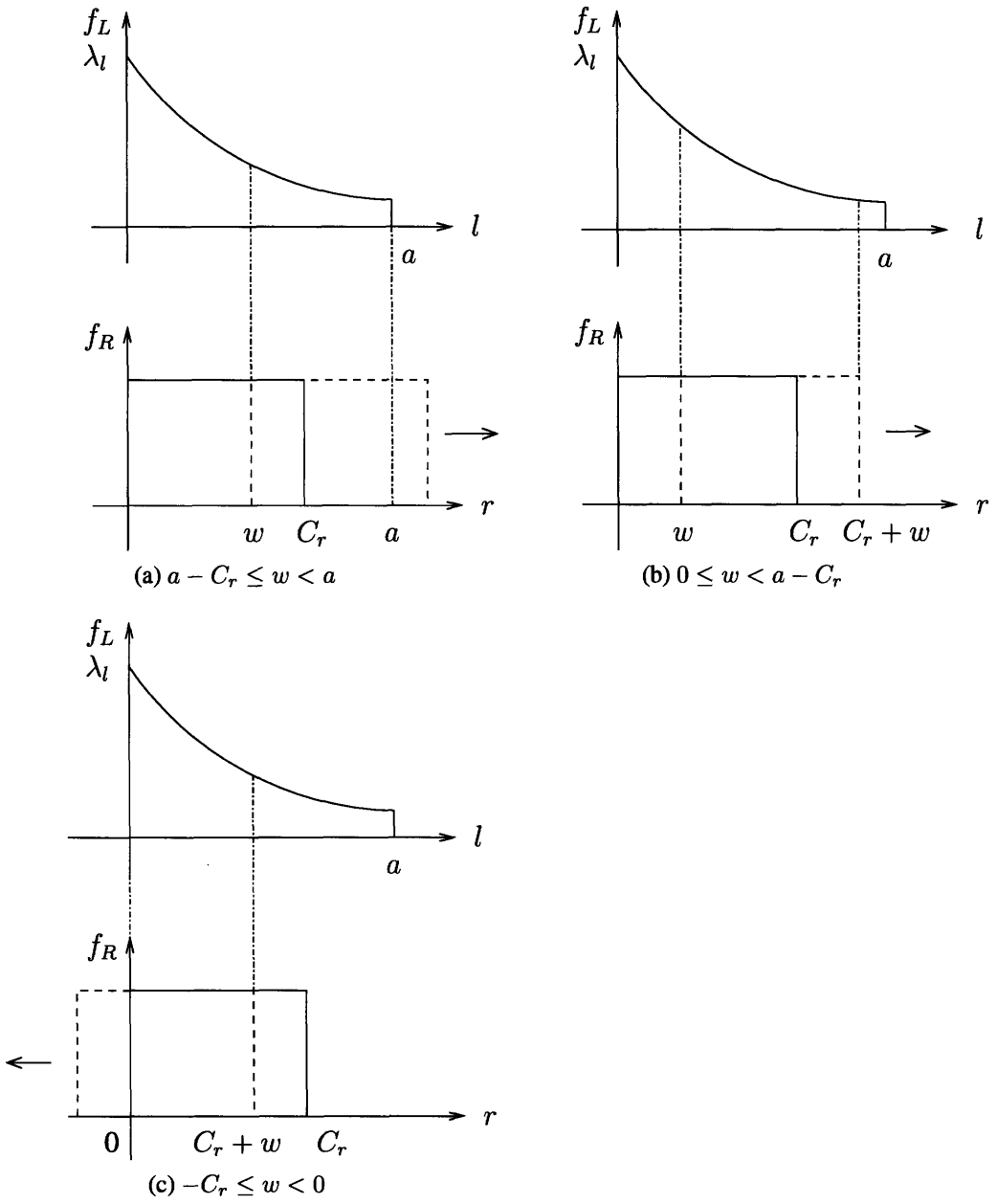


Figure A-2: Convolution in case B. The dashed rectangle shows the shift of f_R due to w.

Bibliography

[1] *The Emulab - Network Emulation Testbed.* http://www.emulab.net/.

[2] *The General Hidden Markov Model library (GHMM).* http://ghmm.sourceforge.net/.

[3] *The Skitter.* http://www.caida.org/Tools/Skitter.

[4] A passive system for server selection within mirrored resource environments using as path length heuristics. Technical Report Technical Report, Applied Theory Communications, Inc., June 1999.

[5] Proximity neighbor selection in tree-based structured p2p overlays. Technical Report Technical Report MSR-TR-2003-52, 2003.

[6] Server-centric view of internet performance: Analysis and implications. Technical Report Technical Report MSR-TR-2001-78, Microsoft Research, September 2007.

[7] Alfarez Abdul-Rahman and Stephen Hailes. Supporting trust in virtual communities. In *Proceedings of HICSS '00*, 2000.

[8] A. Adams, J. Mahdavi, M. Mathis, and V. Paxson. Creating a scalable architecture for internet measurement. In *Proceedings of INET*, 1998.

[9] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The design and implementation of an intentional naming system. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 186–201, New York, NY, USA, 1999. ACM.

[10] Vinay Aggarwal, Anja Feldmann, and Christian Scheideler. Can isps and p2p users cooperate for improved performance? *SIGCOMM Comput. Commun. Rev.*, 37(3):29–40, 2007.

[11] John Mark Agosta, Carlos Diuk-Wasser, Jaideep Chandrashekar, and Carl Livadas. An adaptive anomaly detector for worm detection. In *Proceedings of Second Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML07)*, 2007.

[12] Akamai. http://www.akamai.com/.

[13] Jeannie Albrecht, Ryan Braud, Darren Dao, Nikolay Topilski, Christopher Tuttle, Alex C. Snoeren, and Amin Vahdat. Remote control: Distributed application configuration, management, and visualization with plush. In *USENIX Large Installation System Administration Conference (LISA)*, 2007.

[14] Mark Allman, Ethan Blanton, and Vern Paxson. An architecture for developing behavioral history. In *Proceedings of SRUTI*, 2005.

[15] Mark Allman, Ethan Blanton, Vern Paxson, and Scott Shenker. Fighting coordinated attackers with cross-organizational information sharing. In *Proc. of ACM SIGCOMM HotNets Workshop*, 2006.

[16] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Symposium on Operating Systems Principles*, pages 131–145, 2001.

[17] ARIN. http://www.arin.net/.

[18] Robert Axelrod. *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton University Press, Princeton, New Jersey, 1997.

[19] Hitesh Ballani and Paul Francis. Conman: a step towards network manageability. *SIGCOMM Comput. Commun. Rev.*, 37(4):205–216, 2007.

[20] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani. Estimating aggregates on a peer-to-peer network. In *Submitted for publication*, 2003.

[21] Terry Benzel, Robert Braden, Dongho Kim, Clifford Neuman, Anthony Joseph, Keith Sklower, Ron Ostrenga, and Stephen Schwab. Design, deployment, and use of the deter testbed. In *DETER: Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*, pages 1–1, Berkeley, CA, USA, 2007. USENIX Association.

[22] Dimitri P. Bertsekas and John N. Tsitsiklis. *Introduction to Probability*. Athena Scientific, Nashua, NH, second edition, 2002.

[23] BitTorrent. http://www.bittorrent.com.

[24] Eric Bonabeau. Agent-based modeling: methods and techniques for simulating human systems. *Proc. National Academy of Sciences*, 99(3):7280–7287, 2002.

[25] David Brumley, James Newsome, Dawn Song, Hao Wang, and Somesh Jha. Towards automatic generation of vulnerability-based signatures. In *the 2006 IEEE Symposium on Security and Privacy*, Washington, DC, USA, 2006.

[26] M. Cai, M. Frank, B. Yan, and R.MacGregor. A subscribable peer-to-peer rdf repository for distributed metadata management. In *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 2005.

[27] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles*, pages 298–313. ACM Press, 2003.

[28] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu. Network tomography: Recent developments. *Statistical Science*, 2004.

[29] Y. Chawathe and M. Seshadri. Broadcast federation: An application-layer broadcast internetwork. In *Intnl. Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2002.

[30] Senthilkumar G. Cheetancheri, John Mark Agosta, Denver H. Dash, Karl N. Levitt, Jeff Rowe, and Eve M. Schooler. A distributed host-based worm detection system. In *LSAD '06: Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*, pages 107–113, New York, NY, USA, 2006. ACM Press.

[31] B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords. In *TR CS0917, Technion*, 1997.

[32] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM*, 45, 1998.

[33] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Measurement and Modeling of Computer Systems*, pages 1–12, 2000.

[34] D. Clark. The design philosophy of the darpa internet protocols. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 106–114, New York, NY, USA, 1988. ACM.

[35] D. Clark, C. Partridge, J. C. Ramming, and J. Wroclawski. A knowledge plane for the internet. In *Proceedings of ACM SIGCOMM 2003*, August 2003.

[36] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46+, 2001.

[37] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for p2p systems, 2002.

[38] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. *SIGCOMM Comput. Commun. Rev.*, 34(4):15–26, 2004.

[39] Michael Dahlin, Bharat Baddepudi V. Chandra, Lei Gao, and Amol Nayate. End-to-end wan service availability. *IEEE/ACM Trans. Netw.*, 11(2):300–313, 2003.

[40] Denver Dash, Branislav Kveton, John Mark Agosta, Eve Schooler, Jaideep Chandrashekar, Abraham Bachrach, and Alex Newman. When gossip is good: Distributed probabilistic inference for detection of slow network intrusions. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 1115–1122, 2006.

[41] The GeoIP Database. http://www.www.maxmind.com/.

[42] DETER. http://www.isi.edu/deter/.

[43] Eric Eide, Leigh Stoller, and Jay Lepreau. An experimentation workbench for replayable networking research. In *4th USENIX Symposium on Networked Systems Design & Implementation*, pages 215–228, 2007.

[44] Sameh El-Ansary, Luc Onana Alima, Per Brand, and Seif Haridi. Efficient broadcast in structured P2P networks. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, 2003.

[45] Nick Feamster. *Proactive Techniques for Correct and Predictable Internet Routing*. PhD thesis, MIT, 2006.

[46] M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *the 2nd Theory of Cryptography Conference (TCC '05)*, Cambridge, MA, 2005.

[47] Robert G. Gallager. *Discrete Stochastic Processes*. Springer, first edition, 1996.

[48] BRITE Internet Topology Generator. http://cs-www.bu.edu/brite/.

[49] Gnutella. http://gnutella.wego.com.

[50] P. Goldsack, J. Guijarro, A. Lain, G. Mecheneau, P. Murray, and P. Toft. Smartfrog: Configuration and automatic ignition of distributed applications. In *HP Openview University Association Conference (HP OVUA)*, 2003.

[51] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A clean slate 4d approach to network control and management. *SIGCOMM Comput. Commun. Rev.*, 35(5):41–54, 2005.

[52] Mark Gritter and David R. Cheriton. An architecture for content routing support in the internet. In *USENIX Symposium on Internet Technologies and Systems*, pages 37–48, march 2001.

[53] Web-Ontology (WebOnt) Working Group. http://www.w3.org/2001/sw/webont/.

[54] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 381–394, New York, NY, USA, 2003. ACM.

[55] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *Proceedings of the SIGCOMM Internet Measurement Workshop (IMW 2002)*, Marseille, France, November 2002.

[56] Y. He, G. Siganos, M. Faloutsos, and S. V. Krishnamurthy. A systematic framework for unearthing the missing links: Measurements and impact. In *USENIX/SIGCOMM NSDI*, April 2007.

[57] Ningning Hu. *Network Monitoring and Diagnosis Based on Available Bandwidth Measurement.* PhD thesis, Carnegie Mellon University, 2006.

[58] Yang hua Chu, Aditya Ganjam, T. S. Eugene Ng, Sanjay G. Rao, Kunwadee Sripanidkulchai, Jibin Zhan, and Hui Zhang. Early experience with an internet broadcast system based on overlay multicast. In *ATEC'04: Proceedings of the USENIX Annual Technical Conference 2004 on USENIX Annual Technical Conference*, pages 12–12, Berkeley, CA, USA, 2004. USENIX Association.

[59] Yiyi Huang, Nick Feamster, Anukool Lakhina, and Jim (Jun) Xu. Diagnosing network disruptions with network-wide analysis. *SIGMETRICS Perform. Eval. Rev.*, 35(1):61–72, 2007.

[60] Tim Hwang. Herdict: a distributed model for threats online. *Network Security*, 2007(8):15–18, 2007.

[61] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking, 2003.

[62] IPERF. http://dast.nlanr.net/projects/iperf/.

[63] Jaeyeon Jung, Rodolfo Milito, and Vern Paxson. On the adaptive real-time detection of fast-propagating network worms. In *Proceedings of DIMVA*, July 2007.

[64] Jaeyeon Jung, Vern Paxson, Arthur W. Berger, and Hari Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *IEEE Symposium on Security and Privacy 2004*, Oakland, CA, May 2004.

[65] Michael P. Kasick, Priya Narasimhan, Kevin Atkinson, and Jay Lepreau. Towards fingerpointing in the emulab dynamic distributed system. In *WORLDS'06: Proceedings of the 3rd conference on USENIX Workshop on Real, Large Distributed Systems*, Berkeley, CA, USA, 2006. USENIX Association.

[66] KaZaA. http://www.kazaa.com.

[67] Joanna Kulik. *Network Monitoring and Diagnosis Based on Available Bandwidth Measurement.* PhD thesis, Massachusetts Institute of Technology, 2004.

[68] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *IEEE FOCS*, pages 364–373, 1997.

[69] George Lee. *CAPRI: A Common Architecture for Distributed Probabilistic Internet Fault Diagnosis*. PhD thesis, MIT, 2007.

[70] Ji Li, Dah-Yoh Lim, and Karen Sollins. Dependency-based distributed intrusion detection. In *the 16th USENIX Security Symposium, the DETER Community Workshop*, Boston, MA, August 2007.

[71] Ji Li and Karen Sollins. Exploiting autonomous system information in structured peer-to-peer networks. In *Proceedings of The 13th IEEE International Conference on Computer Communications and Networks (ICCCN2004)*, October 2004.

[72] M.-J. Lin, K.Marzullo, and S.Masini. Gossip versus deterministic flooding: Low message overhead and high reliability for broadcasting on small networks. Technical Report Technical Report CS1999-0637, University of California, San Diego, 1999.

[73] Xin Liu, Qingfeng Huang, and Ying Zhang. Combs, needles, haystacks: Balancing push and pull for discovery in large-scale sensor networks, 2004.

[74] Charles M. Macal and Michael J. North. Tutorial on agent-based modeling and simulation part 2: how to model with agents. In *Winter Simulation Conference*, 2006.

[75] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iplane: An information plane for distributed services. In *OSDI 2006*, November 2006.

[76] David J. Malan and Michael D. Smith. Host-based detection of worms through peer-to-peer cooperation. In *WORM '05: Proceedings of the 2005 ACM workshop on Rapid malcode*, pages 72–80, New York, NY, USA, 2005. ACM Press.

[77] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of 21st ACM Conf. on Principles of Distributed Computing (PODC'02)*, July 2002.

[78] Gurmeet Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: Distributed hashing in a small world. In *Proceedings of 4th USENIX Symposium on Internet Technologies and Systems*, March 2003.

[79] Joseph McAlerney. An internet worm propagation data model. Master's thesis, University of California, Davis, September 2004.

[80] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation*, 2002.

[81] A. Nakao and L. Peterson. Bgp feed configuration memo. Technical report, Department of Computer Science, Princeton University, February 2003.

[82] Akihiro Nakao, Larry Peterson, and Andy Bavier. A routing underlay for overlay networks. In *SIGCOMM*, 2003.

[83] Akihiro Nakao, Larry Peterson, and Andy Bavier. Scalable routing overlay networks. *SIGOPS Oper. Syst. Rev.*, 2006.

[84] Moni Naor and Udi Wieder. Novel architectures for P2P applications: the continuous-discrete approach. www.wisdom.weizmann.ac.il/uwieder, 2002.

[85] Napster. http://www.napster.com.

[86] Ryan Newton, Greg Morrisett, and Matt Welsh. The regiment macroprogramming system. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 489–498, New York, NY, USA, 2007. ACM.

[87] T.S.E. Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 1:170–179 vol.1, 2002.

[88] K. Obraczka and F. Silvia. Network latency metrics for server proximity. In *IEEE Globecom*, November 2000.

[89] Venkata N. Padmanabhan, Sriram Ramabhadran, and Jitendra Padhye. Netprofiler: Profiling wide-area networks using peer cooperation. In *Proceedings of the Fourth International Workshop on Peer-to-Peer Systems (IPTPS)*, Ithaca, NY, February 2005.

[90] Venkata N. Padmanabhan and Lakshminarayanan Subramanian. An investigation of geographic mapping techniques for internet hosts. *Proceedings of SIGCOMM'2001*, page 13, 2001.

[91] Venkata N. Padmanabhan, Helen J. Wang, Philip A. Chou, and Kunwadee Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 177–186. ACM Press, 2002.

[92] Ben Paechter, T. Baeck, M. Schoenauer, A.E. Eiben, and J. Merelo. A distributed resource evolutionary algorithm machine. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC 2000)*, pages 951–958, San Diego, USA, 2000. IEEE, IEEE Press.

[93] KyoungSoo Park and Vivek S. Pai. Comon: a mostly-scalable monitoring system for planetlab. *SIGOPS Oper. Syst. Rev.*, 40(1):65–74, 2006.

[94] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31, 1999.

[95] Larry Peterson, Andy Bavier, Marc E. Fiuczynski, and Steve Muir. Experiences building planetlab. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 351–366, Berkeley, CA, USA, 2006. USENIX Association.

[96] The PlanetLab. http://www.planet-lab.org/.

[97] M. Portmann and A. Seneviratne. Cost-effective broadcast for fully decentralized peer-to-peer networks. *Computer Communication*, Special Issue on Ubiquitous Computing, Elsevier Science, 2002.

[98] The Region Project. http://krs.lcs.mit.edu/regions.

[99] The Route Views Project. http://www.routeviews.org/.

[100] pssh. http://www.theether.org/pssh/.

[101] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in speech recognition*, pages 267–296, 1990.

[102] Venugopalan Ramasubramanian, Ryan Peterson, and Emin Gun Sirer. Corona: A high performance publish-subscribe system for the world wide web. In *Networked System Design and Implementation*, 2006.

[103] Venugopalan Ramasubramanian and Emin Gün Sirer. The design and implementation of a next generation name service for the internet. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 331–342, New York, NY, USA, 2004. ACM.

[104] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*, 2001.

[105] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-level multicast using content-addressable networks. In *3rd International Workshop on Networked Group Communication*, November 2001.

[106] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Topologically-aware overlay construction and server selection. In *INFOCOM'02*, New York, NY, June 2002.

[107] Y. Rekhter and T. Li. A border gateway protocol 4 (bgp-4), 1995.

[108] Martin Roesch. Snort - lightweight intrusion detection for networks. In *LISA '99: the 13th USENIX conference on System administration*, 1999.

[109] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[110] Antony I. T. Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Networked Group Communication*, pages 30–43, 2001.

[111] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, 1984.

[112] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[113] Stephen Schwab, Brett Wilson, Calvin Ko, and Alefiya Hussain. Seer: a security experimentation environment for deter. In *DETER: Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*, pages 2–2, Berkeley, CA, USA, 2007. USENIX Association.

[114] Vyas Sekar, Yinglian Xie, David A. Maltz, Michael K. Reiter, and Hui Zhang. Toward a framework for internet forensic analysis. In *Proc. of ACM SIGCOMM HotNets Workshop*, 2004.

[115] The PLUTO BGP Sensor. http://lurch.cs.princeton.edu.

[116] Georgos Siganos, Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. Power laws and the as-level internet topology. *IEEE/ACM Trans. Netw.*, 11(4):514–524, 2003.

[117] Karen Sollins. Designing for scale and differentiation. In *ACM SIGCOMM 2003 FDNA Workshop*, 2003.

[118] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *Proc. of IEEE INFOCOM 2003*, 2003.

[119] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to own the internet in your spare time. In *the 11th USENIX Security Symposium*, 2002.

[120] Malgorzata Steinder and Adarshpal S. Sethi. Multi-domain diagnosis of end-to-end service failures in hierarchically routed networks. In *NETWORKING*, pages 1036–1046, 2004.

[121] Malgorzata Steinder and Adarshpal S. Sethi. A survey of fault localization techniques in computer networks. *Sci. Comput. Program.*, 53(2):165–194, 2004.

[122] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[123] Lakshminarayanan Subramanian, Sharad Agarwal, Jennifer Rexford, and Randy H. Katz. Characterizing the internet hierarchy from multiple vantage points. In *Proc. of IEEE INFOCOM 2002, New York, NY*, June 2002.

[124] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *ACM SIGCOMM*, 2002.

[125] A. Wald. *Sequential Analysis*. J. Wiley and Sons, New York, 1947.

[126] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An information plane for networked systems, 2003.

[127] Kamin Whitehouse, Cory Sharp, Eric Brewer, and David Culler. Hood: a neighborhood abstraction for sensor networks. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 99–110, New York, NY, USA, 2004. ACM.

[128] M. Wood and M. Erlinger. Intrusion detection message exchange requirements. RFC 4766, IETF, March 2007.

[129] Z. Xu, M. Mahalingam, and M. Karlsson. Turning heterogeneity into an advantage in overlay routing. *INFOCOM 2003. IEEE*, 2:1499–1509 vol.2, 30 March-3 April 2003.

[130] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. Planetseer: Internet path failure monitoring and characterization in wide-area services. In *Proc. Sixth Symposium on Operating Systems Design and Implementation*, December 2004.

[131] Rongmei Zhang and Y. Charlie Hu. Borg: A hybrid protocol for scalable application-level multicast in peer-to-peer networks. In *The 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Monterey, California, 2003.

[132] Zheng Zhang, Shu-Ming Shi, and Jing Zhu. Somo: Self-organized metadata overlay for resource management in p2p dht. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, 2003.

[133] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[134] Ben Y. Zhao, Yitao Duan, Ling Huang, Anthony D. Joseph, and John Kubiatowicz. Brocade: Landmark routing on overlay networks. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 34–44, London, UK, 2002. Springer-Verlag.

[135] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 11–20. ACM Press, 2001.